Bachelor Thesis

about

A Comparison of Open Source and Commercial Solutions for Document Classification

Submitted to:	Faculty of Computer Science and Electrical Engineering of the University of Rostock
Submitted from:	Goergen, Léon Hermann
Student number:	218204406
Course of study:	Business Informatics (B.Sc.)
First supervisor:	Prof. Dr. Kurt Sandkuhl
Second supervisor:	Leon Griesch

Rostock, August 1, 2023

Table of Contents

List of Figures

List of Tables

1. Introduction

Automatic text classification has been a subject of interest since the 1960s and continues to be an important area of study in Natural Language Processing (NLP) and machine learning [1]. The process involves using algorithms to assign labels or categories to text documents based on their content, to minimize manual labor and reduce the likelihood of errors introduced by human factors such as fatigue, lack of expertise, and subjective biases [2]. In the era of big data analytics, where companies often deal with large volumes of data, manual classification of documents can be time-consuming and prone to errors. Therefore, text classification has become essential for businesses and organizations to handle incoming documents effectively and efficiently. This thesis focuses on document classification specifically, which involves assigning a label or category to a document based on its content. It is commonly used for tasks such as spam or fraud detection and organizing incoming documents into classes within a company. [3].

The main objective of this study is to introduce and evaluate a selection of promising Free Open-Source Software (FOSS) approaches for document classification and compare them against each other regarding their prediction performance and efficiency to identify the best candidate for a specific use case. In addition, the study compares the selected approach's prediction performance, efficiency, and cost-effectiveness with commercial providers' proprietary software. This comparison can provide insights into different approaches' relative strengths and weaknesses to help businesses decide on the best strategy for their needs. However, it is essential to note that this thesis focuses solely on a specific use case, and therefore, the results presented may not be generalizable to other use cases.

To facilitate understanding of the study, section 2 provides an overview of the field of document classification, including its fundamental concepts, various data preparation methods, and different FOSS approaches. Section 3 discusses the dataset used, the preprocessing steps taken, the implementation techniques for the FOSS, an overview of proprietary software, and the evaluation metrics used for comparison. Further, section 4 presents the prediction performance and efficiency of the different FOSS approaches on the dataset, providing an analysis of the strengths and weaknesses of each method and how they compare to each other, followed by a comparison of the prediction performance, efficiency, and cost-effectiveness of the best FOSS approach with that of proprietary software solutions. Finally, section 5 summarizes the study's main findings and suggests future research.

2. Overview of the Field of Document Classification

Document classification, a subfield of NLP, is the process of predicting group membership for data instances, meaning organizing documents into predefined categories or classes based on their content [4]. This prediction is often made using machine learning techniques, which involve training a model on a large dataset of labeled documents and then using the trained model to predict classes on new, unseen documents [5]. Document classification is used for various applications, including spam filtering, sentiment analysis, and topic modeling [3]. It is an active area of research with many open problems and opportunities for further study [5].

Section 2.1 will provide a brief overview of the current state of research in document classification. Subsequently, this thesis will introduce the fundamental concept of document classification in section 2.2, which will outline this study. The following sections will discuss each step of the classical workflow, including data preparation techniques in section 2.3, textual preprocessing methods in section 2.4, standard feature extraction techniques in section 2.5, and various approaches for automating document classification using machine learning in section 2.6. These approaches range from traditional methods that dominated the field from the 1960s until the 2010s [2] to deep learning techniques, which have contributed to the development of the most current state-of-the-art systems, such as "DocBERT" [6]. It is worth noting that rule-based approaches were commonly used in the early stages of automated document classification [2]. However, these approaches will not be considered for this discussion, as their performance has not been able to keep pace with more modern technology [2].

2.1. State of Research

The most fundamental and crucial task in NLP is text classification [2], which evolves as new methods and techniques are created and improved. Many approaches, datasets, and evaluation criteria have been proposed in the literature over the past ten years due to the remarkable success in this sector [2]. Recent advances in machine learning and NLP have led to significant progress [3], such as deep learning techniques, which have shown promising results in improving the accuracy of document classification models [2]. In addition, transfer learning has emerged as a popular approach in document classification, in which Transformer-based Pre-trained Language Models (PLMs), such as Bidirectional

Encoder Representations from Transformers (BERT), are trained on a vast document corpus and fine-tuned on a specific document classification task [7]. This method has achieved state-of-the-art results on many benchmark datasets, such as the "IMDB movie review dataset", the "20 Newsgroups dataset", and the "Reuters-21578 dataset" [2, 3].

Furthermore, research in document classification has focused on addressing some of the challenges in this field, such as the high dimensionality of document features, which can lead to computational complexity and overfitting [5, 8]. To address this challenge, researchers have explored techniques such as dimensionality reduction, feature selection, and feature engineering [8]. Another challenge is the issue of class imbalance, where some categories have significantly fewer documents than others, leading to biased models [9]. Researchers have proposed various techniques to address this challenge, such as over-sampling, under-sampling, and class weighting [9]. Future research will likely address the remaining challenges in this field, such as the interpretability of models, the incorporation of external knowledge, and the development of models that can handle multiple languages and domains [2].

2.2. Fundamental Concept

The fundamental concept of automated document classification is to classify documents into predefined categories or labels in a computerized manner based on the characteristics of the documents and the relationships between them [5]. Document classification aims to accurately and efficiently categorize large volumes of documents to facilitate information organization, search, and retrieval [5]. Therefore, researchers follow joint guidance, which has proven efficient [5]. This section presents the fundamental concept of utilizing machine learning techniques to achieve this specific goal, starting with an introduction to the general workflow to any document classification task in section 2.2.1, including a brief introduction to each step, which sections 2.3 to 2.6 will explain in more detail. Additionally, section 2.2.2 introduces various machine learning paradigms, which can be utilized to classify documents effectively.

2.2.1. Workflow

In most cases, the general workflow for document classification follows a typical pattern [5], as illustrated in Figure 1, which will also serve as the outline for this thesis.

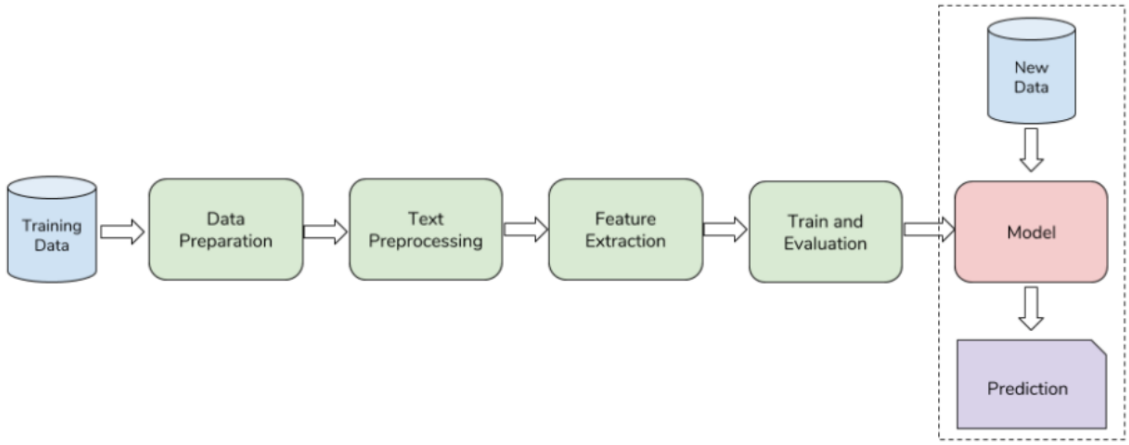


Figure 1: Text Classification Workflow [10]

Before the training process begins, it is necessary to prepare the given training data [9, 11]. This preparation typically involves removing redundant or irrelevant samples or features, detecting and addressing outliers that could affect the output, and balancing the dataset to avoid introducing bias [9]. These steps are essential to ensure the training process is reliable and accurate [12]. It is worth noting that the training data may be provided in an undesirable format, such as PDF files, requiring techniques such as Optical Character Recognition (OCR) to extract the content. After extracting and formatting the training data as needed, most studies, such as [13], recommend preprocessing the textual content using techniques such as stop word removal, spelling correction, and lemmatization. These preprocessing steps can be very beneficial in improving the accuracy of the document classification system [14].

The following step, "feature extraction", is a process that involves reducing the data to its most essential features, with intending of discovering the underlying structure of the data [15]. Then, the model can interpret and use the data as input by converting the text into numerical representations, such as word embeddings [15]. Often, feature extraction follows "feature selection", which involves selecting a subset of the most relevant and informative features from the dataset to be used as input to the model while discarding the rest [16]. Feature selection can help reduce the feature set's dimensionality and improve the model's efficiency and performance, mainly when working with large datasets [17]. While feature

extraction and selection are crucial in many traditional machine-learning pipelines, they are often automated or handled implicitly in deep-learning models [2]. Deep learning models can learn features directly from the data without manual feature engineering, which allows them to capture complex patterns and relationships in the data and can lead to improved performance on many tasks [2].

The final step in the machine learning workflow involves setting up the desired model, selecting appropriate hyperparameters, and training the model on the preprocessed and feature-engineered data [5]. Once the model is prepared, it is vital to evaluate its performance on the test data and, if necessary, adjust the hyperparameters to optimize its implementation [18]. After the model has been trained or fine-tuned, it can predict classes on new, unseen data in production.

2.2.2. Machine Learning Paradigms

This section provides an overview of the different types of machine learning and how they can be applied. Figure 2 provides a visual summary of the various machine-learning approaches and how they relate. The different types of machine learning can be broadly categorized into supervised, unsupervised, and reinforcement learning, depending on the nature of the input data and the desired output [16]. This thesis primarily focuses on supervised classification tasks; however, to provide a general overview of machine learning, this thesis briefly explains the paradigms based on the comprehensive explanation of [16].

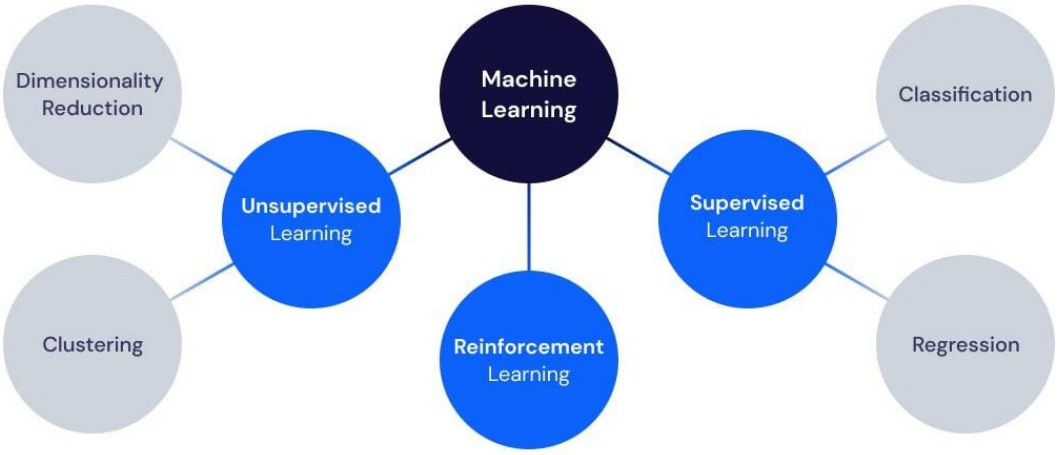


Figure 2: Machine Learning Paradigms [19]

Supervised Learning

In supervised learning, the machine learning engineer trains a model using a labeled dataset which provides the correct output label for each example. The training aims to infer a function that maps new, unseen data points to a desired output pattern [20]. As supervised learning tries to achieve a specific goal, it is considered a *task-driven* process [21]. Supervised machine learning can be divided into two tasks: "Regression" describes the prediction of a result variable of a continuous quantity based on one or more input variables [16]. Typical applications are financial forecasting, cost estimation, or trend analysis. In "classification" tasks, the model should predict a predefined class label for a given input [20]. Based on the number of classes and the number of affiliations, there are three categories:

- **Binary Classification**: Binary classification refers to tasks where there are only two possible class labels, such as "true" or "false", or in the case of spam detection, "spam" or "not spam" [20].
- **Multi-class Classification**: In multi-class classification, the goal is to assign a single class label to each input example, where the number of possible class labels is higher than two, contrasting binary classification, where there are only two possible class labels [20]. An example of a multi-class classification task is identifying a news story belonging to one of several possible classes, such as a business, politics, or sport. This type of classification will be the focus of the study.
- **Multi-label Classification**: Multi-label classification has at least three possible class labels, but each example can belong to multiple classes [22]. An example of a multi-label classification task is the classification of movies, where a single film can belong to various genres simultaneously.

Unsupervised Learning

Unsupervised learning is a machine learning paradigm that involves analyzing an unlabeled dataset without the guidance of human supervision [20]. Instead, unsupervised learning algorithms use the data's inherent structure to identify patterns and relationships, making it a *data-driven* process [20]. This approach is practical when labeled data is unavailable or difficult to obtain, as it allows the algorithm to learn from the data itself.

There are two most common types of unsupervised learning [16]:

- **Clustering** or **Cluster Analysis** is the process of dividing a dataset into groups (called clusters) based on the patterns or relationships among the data points without concern for the specific outcome and is frequently employed as a method of data analysis to find intriguing trends or patterns in data [16, 20]. Some common approaches are "partitioning", "density-based", and "hierarchical-based" methods [16].

- **Dimensionality reduction** is a technique for reducing the number of variables in a dataset by extracting meaningful information and discarding less critical information [16]. It addresses the problem of high-dimensional data, which can be challenging to process and visualize, by simplifying the features in the dataset [16]. There are two types of dimensionality reduction techniques: In "feature selection" methods, a subset of distinctive characteristics is selected from the dataset, reducing the complexity of the model by removing the unnecessary or unimportant features and enabling quicker training [16, 23]. In contrast, feature extraction aims to decrease the number of features in a dataset by creating new ones from the ones that already exist and then discarding the original features [24].

Semi-Supervised

Semi-supervised learning is a machine learning approach that involves training a model using a combination of labeled and unlabeled data and is, therefore, a hybridization of the methods above [20, 21]. Mostly there is a large amount of unlabeled data and only a limited amount of labeled data, so semi-supervised machine learning can effectively improve model performance by leveraging the large amounts of unlabeled data [25]. However, it can be challenging to define the specific conditions under which a particular semi-supervised learning algorithm will be effective, and it is often difficult to evaluate the extent to which these conditions are satisfied in a given context. As a result, it can be challenging to identify specific use cases for which semi-supervised learning is particularly well-suited [26].

Reinforcement Learning

Thanks to the machine learning technique known as reinforcement learning, software agents and robots can automatically assess the best behavior in a given context or environment. Therefore, this is an *environment-driven* approach [27]. This learning strategy directs the agent's decision-making based on incentives and punishments. Reinforcement learning uses the knowledge gained from environmental feedback to raise the reward or lower the risk associated with a given behavior [25]. In reinforcement learning, the agent interacts with the environment and learns from its responses by being rewarded or penalized. For the agent to learn from its experiences and make better decisions moving forward, this input is utilized to update the agent's policy or decision-making strategy. The agent can discover the behaviors that yield the most significant rewards through trial and error and reduce the behaviors that yield the most significant penalties [25].

Reinforcement learning is a powerful method for training AI models to increase automation or boost the operational efficiency of intricate systems like manufacturing, supply chain logistics, autonomous vehicles, and robotics [16]. However, for straightforward issues, such as document classification, it might not be the ideal strategy as the difficulty of the procedure can occasionally outweigh the advantages [16].

2.3. Data Preparation

Implementing machine learning models presents various challenges, among which ensuring adequate data quality is considered one of the most significant, as the performance and success of the classifier heavily rely on this quality [11]. Generally, the selection of data preparation methods for a particular use case differs, resulting in a non-reproducible data preprocessing pipeline, where preparation is performed manually and individually [9]. However, machine learning engineers can adapt fundamental data preparation concepts to almost every machine learning composition. This section provides an overview of the most crucial steps in the data preparation pipeline for textual datasets, including data cleansing in section 2.3.1 and data augmentation and balancing in section 2.3.2. The reader should refer to [9, 11] for a more comprehensive and detailed explanation.

2.3.1. Data Cleansing

Data cleaning is often necessary, as missing values, outliers, and noisy data are prevalent in most real-world production data sets, impairing data analysis performance and reducing efficiency [9]. Ignoring, deleting, or imputing missing data is possible [9]. Unbiased modeling results from ignoring missing data, although engineers should consider this technique only when the percentage of missing values is low [9]. If there are enough instances or attributes to avoid losing too much information, eliminating data entries with missing values via deletion can be considered. Imputation is the most popular method for addressing missing values because it preserves relevant information, especially if only a few data entries are available [9]. Outliers, categorized as global, contextual, and collective outliers, are exceptional values that differ from other observations and can harm modeling [9]. Controlling outliers is similar to handling missing data since engineers can disregard, eliminate, or impute outliers [9]. It is also necessary to eliminate duplicate features, constant values, and correlations between components because these characteristics do not add meaningful details for modeling and might negatively impact the models' performance [9].

2.3.2. Balancing and Augmenting the Dataset

It is typical for classes to be distributed inconsistently in classification tasks, and this class under-representation provides a substantial issue because algorithms benefit from balanced data [11]. Algorithms may fail to recognize patterns in severe under-representation and instead generate predictions based solely on the output class that represents the majority, leading to bias against one class [11]. To address the problem of imbalanced datasets, the most common solutions are under-sampling, which means simply downsizing the majority classes by removing data entries, and over-sampling, which describes the process of creating synthetic samples from the minority classes [9, 11]. Machine learning engineers can use data augmentation, which is closely related to over-sampling, to expand the data set and increase its variance if the whole dataset contains too few samples to train a classifier adequately. New features can be derived from existing ones, giving machine learning models access to further, meaningful information. Augmentation can be accomplished, for example, by joining many columns into one or creating synthetic samples by altering the original text [9].

2.4. Textual Preprocessing

Text documents usually contain various unwanted elements, such as slang, incorrect spelling, and punctuation [28]. Therefore, preprocessing techniques are beneficial for reducing data inadequacy and sparsity to improve the overall quality of text data [28]. In addition, research has demonstrated that "appropriate text preprocessing, including data transformations and filtering, can significantly enhance the classifier's performance" [13]. Below, this section discusses some standard preprocessing methods in detail:

- **Tokenization** Tokenization is segmenting a text into separate units, commonly referred to as tokens, which can be words or sequences of words [29]. The primary purpose of tokenization is to identify meaningful words or phrases in a sentence [30].

- **Removal of Punctuation and non-ASCII Characters** In specific scenarios, removing non-alphabetic characters from the text is suggested to enhance the statistical power, which includes special characters absent in the American Standard Code for Information Interchange (ASCII) symbols, as well as punctuation such as exclamation marks, question marks, dots, commas, and quotation marks [14].

- **Correction of Spelling mistakes** Incorrect spellings within a document can increase vocabulary size, negatively impacting the system's efficiency and decreasing statistical power [14]. To mitigate spelling mistakes, engineers can deploy spelling correction tools to standardize the text and eliminate noise [14].

- **Expanding Contractions and Abbreviations** Similar to incorrect spellings, using contractions and abbreviations within a document may increase vocabulary size, harming the overall statistical power [14].

- **Lowercasing all words** Standardizing the case of characters within a text document, specifically by converting all characters to lowercase, can improve the system's overall accuracy, reducing the overall vocabulary size [14].

- **Removing Stopwords** Stopwords can be defined as words that frequently appear in a document but do not contribute meaningfully to the content [31]. These stopwords can be specific to a language (e.g., for English, words such as "I" or "am") or related to a particular topic [31].

- **Stemming** A commonly employed technique in information retrieval is the process of stemming text to its morphological roots, resulting in reduced vector sizes [32]. However, it is debatable whether stemming improves text classification performance, as research by [33] has shown.

- **Lemmatization** Lemmatization is a process that aims to reduce words to their base or dictionary form using lexical knowledge, much like stemming [34]. However, while stemming removes inflectional endings, lemmatization considers the context and vocabulary of the words. Research has indicated that lemmatization can achieve higher precision than stemming in text classification tasks [34].

2.5. Feature Extraction

Feature extraction enables machines to classify and comprehend data like humans and involves converting raw data into numerical representations that models can interpret and process [28]. Through the years, research has proposed various algorithms to address the problem of syntactic and semantic information loss that occurs when converting words to numerical values [28]. In a machine learning-based model or system, selecting the appropriate feature extraction technique is crucial since different approaches offer a better comprehension of the data, a way to increase prediction accuracy. In contrast, others reduce computing costs or training time [16]. The right choice depends, therefore, on the specific use case. The following sections discuss commonly employed methods.

2.5.1. Classical Representations

This section will examine standard classical models that were frequently applied to text classification in the past [28]. These methods rely on turning words into vectors, including the number of words that exist in a specific document and representing terms based on their frequency [28]. Classical word representations can be divided into two groups [28], which will be briefly discussed below:

Categorical word representations techniques convert words into discrete categorical variables by encoding them as integers, representing the word's position in a fixed vocabulary [28]. One of the main benefits of categorical word representations is that they are easy to work with and can be incorporated into many different machine-learning models [28]. Under categorical word representation methods, there are two models:

- **One-hot encoding** One-hot encoding is a method of representing text data as a numerical vector by creating a binary vector of the same length as the vocabulary and setting the value of the element corresponding to the word in the vocabulary to "1" while setting the values of all other elements to "0" [28].

- **Bag-of-Words (BOW)** represents a text document in a formal extension of the one-hot encoding method, which is achieved by summing up the one-hot encodings of each word in the text document element-wise [28]. This results in a single vector representing the text document, where the value of each element corresponds to the number of occurrences of the corresponding word in the document [35].

Since these techniques concentrate on producing a numerical representation of the text based on the frequency or presence of words rather than capturing the underlying meaning of the words, neither one-hot encoding nor BOW captures the semantic meaning of the words nor the context and relationships between words in the resulting representations [35]. Table 1 represents an example of one-hot encoding and BOW. The table illustrates the use of these techniques in the sample text "the cat sat on the mat". The dictionary used in this example consists of all unique words in the sample text and an additional term, "dog", for demonstration purposes. The table presents the one-hot encodings for each word in the sample text and the element-wise addition of these encodings to obtain the BOW representation for the sample text.

		dictionary					
		cat	dog	mat	on	sat	the
input	the	0	0	0	0	0	1
	cat	1	0	0	0	0	0
	sat	0	0	0	0	1	0
	on	0	0	0	1	0	0
	the	0	0	0	0	0	1
	mat	0	0	1	0	0	0
BOW		**1**	**0**	**1**	**1**	**1**	**2**

Table 1: Example of One-Hot-Encoding and BOW

Weighted word representation methods create numerical representations of text based on the frequency of words in the text by assigning a weight to each word, with the weight being a measure of the importance or relevance of the word in the text [28]. This technique allows for more informative and expressive representations of the text, as it captures the relative importance of words rather than only their presence or absence, as seen in the categorical approaches [36].

- **Term Frequency (TF)** is a statistical measure that quantifies the frequency of a given word within a document by assigning a numerical value that indicates the number of occurrences of a word in the document, normalized by the total number of words in the document [28, 37]. Therefore, this measure considers words that appear more frequently in a document to be more important or relevant to the document's meaning [37]. This normalization step is practical when comparing term frequencies across documents of different lengths [37]. The calculation of the TF is straightforward:

$$TF(w, d) = \frac{occurrences\ of\ term\ t\ in\ document\ d}{total\ number\ of\ terms\ in\ document\ d} \tag{1}$$

- **Term Frequency-Inverse Document Frequency (TF-IDF)** is a combination of the previously discussed TF and Inverse Document Frequency (IDF), which measures how rare a word is across a corpus of documents, with the idea that words common across all documents are less informative than rare words [37]. Therefore, this weighting scheme assigns a high value to words that frequently occur in a specific document and rarely in the whole corpus, which are considered more informative and essential for the representation of the document [37]. The TF-IDF weight of a word in a document is computed as the product of its TF and IDF as follows:

$$TF - IDF(t, d, D) = TF * IDF = TF * \log\left(\frac{D}{df_t}\right) \tag{2}$$

  Whereas t denotes the terms, which are the unique words in the text corpus, d denotes each document in the corpus, D represents the collection of all documents in the corpus, and df_t denotes the number of documents that contain the term t [28].

- **Latent Semantic Analysis (LSA)**, first introduced by Dumais, Furnas, Landauer, and Deerwester in 1988 [38], is a powerful mathematical method used to uncover the underlying structure of a set of documents, with the principle that words with similar contextual usage tend to have similar meanings [39]. As a result, the method employs these words to extract meaning from unstructured text data. LSA can identify latent (hidden) relationships between words and documents by utilizing Singular Value Decomposition (SVD) and is, for that reason, able to find patterns and relationships that might not be immediately obvious [38]. The basic process consists of first creating a document-term matrix (with, e.g., BOW or TF-IDF), which is followed by a decomposition using SVD, which is a mathematical technique for identifying latent relationships between the rows and columns of the matrix:

$$A = U * \Sigma * V^T \tag{3}$$

  in which A is the matrix to decompose, U is the matrix of left singular vectors, which represents the relationships between the documents in the new latent semantic space, Σ is the matrix of singular values, which denotes the importance of each dimension in the latent semantic space, and V^T is the matrix of right singular vectors, which represents the relationships between the terms in the new latent semantic space [40]. The results of the SVD are then used to reduce the dimensionality of the matrix by keeping only the most important latent relationships, which will return a new matrix of latent semantic space, representing each document and each term by a set of coordinates in this space. The similarity between the two data points is then measured by the distance between the coordinates of the two in this space [38, 41].

2.5.2. Feature Learning

As described above, NLP and machine learning extensively utilized category and weighted word representations [28]. However, these approaches suffer from several limitations, including their inability to capture the syntactic and semantic nuances of words and the issue of high dimensionality [28]. Consequently, researchers have explored using distributed word representations in a low-dimensional space, developing word embeddings [42]. The process of word embedding involves transforming alphanumeric characters into vector shapes, where a vector represents each word, denoting a specific dimension of a point in space, which allows for clustering words with shared features, such as contextual

or semantic similarity, in close proximity within the space [43, 44]. While it is possible to train custom word embeddings while implementing a machine learning model, various word embedding models were pre-trained on large corpora of text documents. As a result, these pre-trained models can be used to generate high-quality word embeddings that can improve the performance of various NLP tasks [28]. Below are some commonly used pre-trained word embedding models:

- **Word2vec** was initially proposed by Mikolov et al. in 2013 [45]. It creates a vector of each word using two hidden layers in a shallow neural network, which allows it to capture a word's semantic and grammatical information [28, 44]. There are two different algorithms in word2vec: Continuous Bag of Words (CBOW) and Skip-gram. CBOW predicts the target word given some context words, Skip-gram predicts the context words given the target word, while a window size declares the number of words before and after the target word, creating the context for the target word [44]. Both algorithms update the internal weights of the model based on how well they can predict the target word or context words [2].

- **Global Vectors for word representation (GloVe)** Word2vec is a popular neural network model that generates high-quality word embeddings by focusing on local context window knowledge. However, it needs to make better use of global statistical information. To address this limitation, the GloVe model was proposed by Manning et al. in 2014 [46]. Word co-occurrence in a corpus sets the base idea of GloVe. In the first step, GloVe creates a global co-occurrence matrix from the dataset, which captures the number of times each word appears in the context of every other word in the corpus. In the second step, GloVe uses factorization to obtain the word embeddings. By incorporating local context and global statistical features, GloVe produces word embeddings that capture the syntactic and semantic relationships between terms from both the local context and the global appearances of words in the corpus [2, 28].

- **Embedding from Language Models (ELMo)** is a deep contextualized word embedding model introduced in 2018 by researchers at the Allen Institute for Artificial Intelligence [47]. Unlike traditional word embedding models, such as word2vec and GloVe, ELMo incorporates contextual information into the embeddings [28]. Traditional word embeddings generate a fixed vector representation for each word in the vocabulary, regardless of its context. In contrast, ELMo generates a dynamic embedding for each word, which varies depending on its context in the sentence. Learning from bi-directional language models, which consider the surrounding words on both sides of the target word, achieves this embedding [28]. A typical example would be the word "bank" occurring in different contexts, such as "river bank" and "bank account". Traditional word embedding models like word2vec and GloVe would generate a static word representation with the same vector for both contexts. However, ELMo generates a dynamic embedding for each occurrence of the word that varies depending on its context in the sentence. Thus, in the case of "bank", ELMo would assign different values for its embeddings based on its usage in other contexts, while word2vec and GloVe would give the same value to its embeddings regardless of context [28].

In addition to the previously mentioned word embedding models, most transformer models have unique word embedding techniques, as discussed in section 2.6.2. These techniques address each transformer model's challenges, requirements, and NLP tasks.

2.6. Document Classification Models

This section will explore various, widely renowned, and most frequently used machine learning approaches to document classification, including traditional machine learning methods such as Naive Bayes Classifiers (NBCs), K-Nearest Neighbor (KNN) algorithms, Decision Trees, Logistic Regression, and Support Vector Machines (SVMs) in section 2.6.1, which are based on statistical models that learn from labeled data. Additionally, section 2.6.2 will delve into deep learning models such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformers, which have been widely used in NLP and are effective in handling complex and large-scale text datasets [48]. Finally, the section will discuss each model's working mechanisms, advantages, and limitations in document classification.

2.6.1. Traditional Methods

Traditional, statistic-based models have been the dominant approach to document classification from the early 1960s until the 2010s, building upon the successes of earlier rule-based systems [2]. However, these models require significant feature engineering, including word encoding and feature extraction techniques, to process text data for use in the model [2]. Several traditional classification algorithms have been published in the literature on machine learning and data science; the most popular ones [16] will be discussed below and contrasted in section 3.

Naive Bayes Classifier

NBCs are renowned for being straightforward but highly effective in text classification [49]. Bayes theorem serves as the foundation for the probabilistic model of NBCs. The term "naive" refers to the naive assumption that features in a dataset are independent of one another. Although the independence assumption is somewhat unrealistic NBCs perform well [50, 49]. For the simple NBC, the posterior probability $P(w_j|x_i)$, given a feature vector x of sample i and notation w of several classes j, can be represented using the generic equation below:

$$P(w_j|x_i) = \frac{P(x_i|w_j)P(w_j)}{P(x_i)} \tag{4}$$

whereas $P(w_j)$ reflects the prior probability, or in other words, the general likelihood of encountering a particular class, $P(x_i)$ deflects the probability of encountering a specific pattern x independent from the class label, and $P(x_i|w_j)$ denotes the likelihood of feature x_i belonging to class w_j [49]. As the denominator $P(x_i)$ does not dependent on the class, and under the *naive* assumption that the individual x_i are independent of each other and thus $P(x_i|w_j)$ can be rewritten as

$$P(x|w_j) = P(x_1|w_j) * P(x_2|w_j) * \ldots * P(x_d|w_j) = \prod_{k=0}^{d} P(x_k|w_j) \tag{5}$$

the most probable class $\hat{y}$, also known as *maximum a posteriori*, can be determined as follows [51]:

$$\hat{y} = \arg\max_{y_j} \prod_{k=1}^{p} P(x_k|y_j)P(y_j) \tag{6}$$

while y_j stands for the various classes, $\prod_{k=1}^{p} P(x_k|y_j)$ denotes the likelihood of feature x_i belonging to class w_j for every feature p, and $P(y_j)$ represents the general likelihood of encountering a particular class. Compared to more complex methods, the paramount advantage of NBCs is their ability to estimate the required parameters quickly with a small amount of training data [22]. However, its performance may be compromised due to its strict assumptions of feature independence [22]. The NBC has several common variants, including Gaussian, Multinomial, Complement, Bernoulli, and Categorical [22]. However, considerable research, including [52], has shown that the Multinomial NBC performs better for document classification tasks. For this reason, this version will be employed in this thesis.

k-Nearest Neighbors

The KNN algorithm is a non-parametric classification method and a well-known solution for classification problems, as it has demonstrated simplicity and effectiveness in many scenarios [53, 54]. The algorithm is called "instance-based learning" or non-generalizing learning [16]. It is also commonly known as a "lazy learning" algorithm because it does not involve creating a general internal model. Instead, it simply stores all instances corresponding to the training data in a n-dimensional space [16].

In a training dataset D, the data tuples $d_i \in D$ are represented as vectors in an m-dimensional feature space, where each dimension corresponds to a feature or attribute of the sample. This leads to a representation of each data tuple d_i in the form of $d_i =< w_{i1}, w_{i2}, \ldots, w_{im} >$, in which w_{im} represents the weighted representation of each feature [54]. A distance measure, such as Euclidean distance, Manhattan distance, or cosine similarity, determines how similar the training samples and a new sample are [55]. Based on the distance measure, the KNN method chooses the "k" nearest neighbors from the training dataset [55]. Then, by either taking a majority vote among the "k" nearest neighbors or computing a weighted average of the class labels depending on the proximity of the nearest neighbors to the new sample, the algorithm will predict a class label for the unknown example [55].

The KNN algorithm is relatively resilient to noisy training data, and its accuracy is contingent on the data quality [16]. However, selecting the ideal number of neighbors to be considered presents a significant challenge for KNN as it tremendously impacts prediction accuracy [16, 54]. Moreover, the KNN algorithm's execution time on large datasets is unusually long because of the direct relationship between the model's time/space complexity and the volume of data [56].

Decision Tree

Decision trees have been widely utilized in document classification tasks [16]. It combines a series of simple tests sequentially, resulting in an effective and easy-to-interpret model [57]. The structure of a decision tree consists of nodes that represent features or attributes of the document being classified, with the branches representing the possible values of those attributes. The leaves of the tree correspond to the classes or categories into which the document can be classified [8]. For inference, the algorithm follows the branches of the tree based on the values of the document's attributes until it reaches a specific leaf, indicating the predicted class for the data point [8].

It is simple to explain decision tree classifiers, and their visual representations make them easy to comprehend [28]. Additionally, these classifiers have very few hyperparameters that need to be adjusted [28]. However, experimental evidence indicates that text classification tasks often require considering a significant number of relevant features, which results in the tendency of decision trees to make classifications based on minimal testing, which can lead to unsatisfactory performance in text classification tasks [58].

Logistic Regression

Logistic regression, a statistical method used for classification, has been utilized since early times [28]. However, its significance has grown in recent years, and its usage has significantly increased [53]. The algorithm calculates the sum of every input feature f_i multiplied by its specific weight w_i for the total number of features N. The output of this function is interpreted as the probability of the input data x belonging to the class c [59, 60, 61]:

$$P(c|x) = \sum_{i=1}^{N} w_i f_i \tag{7}$$

As the output ranges from $-\infty$ to $+\infty$ an exponential function ranges that output in the logit scale (e.g., from 0 to 1). Also, the features used to train the model are often properties of the input observation x. However, in some instances, the features may also be dependent on the candidate output class c that is being predicted. Therefore, a feature function $f_i(c, x)$ accounts for this dependency. This function assigns a feature i to the combination of class c and input observation x, enabling the model to capture the relationship between the features and the candidate output class [60, 61]. The previous steps lead to the following equation:

$$P(c|x) = \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c, x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_i f_i(c', x)\right)} \tag{8}$$

in which C represents the total number of classes, and c' denotes an index that ranges over all possible output classes in the set C [60]. Logistic regression is a statistical method that makes predictions based on independent variables. Nevertheless, choosing the wrong independent variables can lead to incorrect predictions [28]. Also, it assumes that each data point is independent; therefore, correlated data might adversely affect the prediction accuracy [28]. Logistic regression is effective on datasets with many dimensions but may overfit rapidly, and one significant limitation is its assumption of linearity between dependent and independent variables [16]. Although logistic regression can be used for both classification and regression tasks, it is typically used for classification [16].

Support Vector Machine

SVMs were first proposed in 1992 by Boser, Guyon, and Vapnik at the Conference on Learning Theory [62]. SVMs have demonstrated high efficacy in the context of pattern classification problems, exhibit scalability to high-dimensional data [63], and are well-known and frequently utilized in text classification tasks, as their performance has proven to be highly effective [28]. Additionally, the trade-off between classifier complexity and error can be controlled explicitly, providing flexibility in the SVM training process [63]. A SVM, at its most basic level, is a binary classifier that seeks out the hyperplane that best divides the data into distinct groups. A hyperplane divides the feature space into two regions for each class. Finding the hyperplane that maximizes the margin, or the separation between the hyperplane and the nearest points (support vectors) from each category, is the objective of a SVM [63, 64].

A general equation for finding the maximum margin hyperplane in a simple linear SVM classifier is given as follows:

$$margin = \arg\min_{x \in D} d(x) = \arg\min_{x \in D} \frac{|x \cdot w + b|}{\sqrt{\sum_{i=1}^{d} w_i^2}} \tag{9}$$

whereas $|x * w + b|$ denotes the distance between the data point x and the decision boundary, which is represented by the hyperplane defined by the weight vector w and the bias term b, d reflects the dimensionality of the input space, and w_i is the i-th component of the weight vector w [63, 64]. SVMs are effective in high-dimensional spaces and can behave differently based on various kernel functions, such as the "sigmoid" function [22], which will be used in this study. However, when the dataset has more noise or overlapping target classes, SVMs do not perform well [16]. Another drawback of SVMs is the challenge of selecting an appropriate kernel function, which is complex and non-trivial [28]. Additionally, SVMs may require lengthy training, especially when dealing with large datasets and consume considerable computational resources [28].

2.6.2. Deep Learning Methods

This section focuses on deep learning models relevant to document classification tasks. Deep learning models have become increasingly popular because they can handle complex NLP tasks [2]. Neural networks are a class of machine learning algorithms inspired by the human brain's structure and function [2, 65]. They are used to recognize data patterns and perform classification, regression, and other tasks that require identifying relationships between variables. As shown in Figure 3, a neural network comprises layers of interconnected nodes, called neurons, which process information. Each neuron takes inputs, applies a mathematical function to them, and produces an output passed on to the next layer of neurons [65]. In addition, the connections between neurons are weighted to enhance the neural network's performance. Finally, the neural network modifies these weights throughout training [65]. As a result, neural networks are highly effective at solving complex problems and have become a central tool in artificial intelligence [2, 65].

The section overviews deep learning models such as RNNs, CNNs, and transformers. RNNs are a class of neural networks that can process sequential data, making them ideal for tasks such as text analysis, sentiment analysis, and speech recognition [2]. Popular

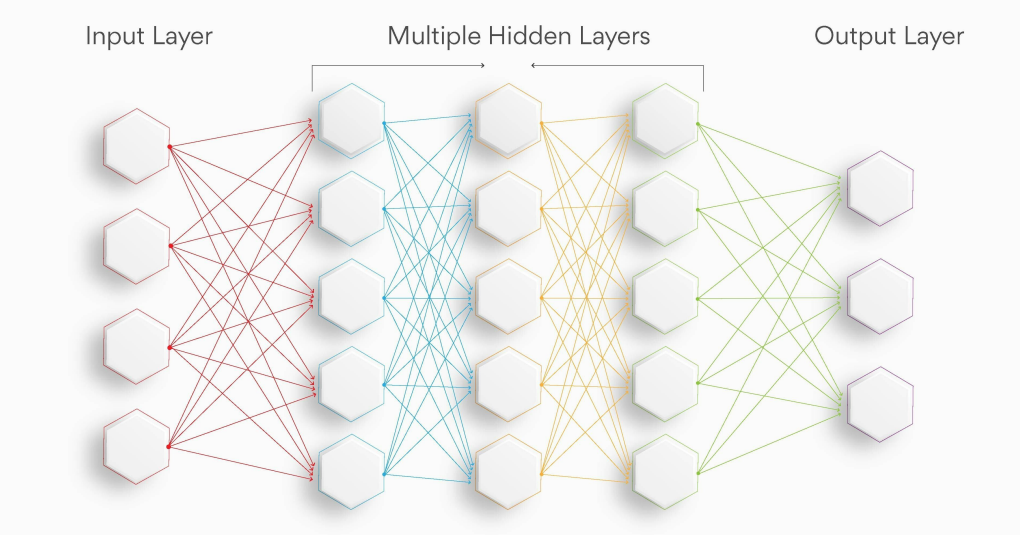


Figure 3: Neural Network Architecture [66]

RNN models such as Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (Bi-LSTM) are discussed, highlighting their strengths and weaknesses. Additionally, CNNs are well-suited for document classification tasks [2], and the section examines their architectures and applications. Finally, the section explores transformers, a relatively new deep learning model that has revolutionized NLP tasks and is considered the current state-of-the-art [67]. This section provides an overview of these deep learning models, their architectures, and their applications in document classification, laying the groundwork for the subsequent analysis presented in this study.

Recurrent Neural Network

RNNs, based on the work of [68] in 1982, are a widely used approach for capturing long-range dependencies in sequential data, including text [48]. RNN language models are trained to learn historical information by considering the order of words in a text [2]. Each input word is first transformed into a vector representation using word embedding technology. These embedding vectors are then sequentially fed into the RNN cells, one by one [2]. The output of the RNN cells, which has the same dimension as the input vector, is then passed into the next hidden layer [2]. The RNN shares parameters across different model parts, meaning that the same weights are used for each input word [2]. In the final step, the output of the last hidden layer is used to predict the label of the input text [2].

RNNs update their weights using gradients by continuously multiplying derivatives during backpropagation [2]. However, if the derivatives become too small, it can cause the problem of vanishing gradients, where the gradients become too small to update the weights effectively [2]. To overcome this problem, an improved version of RNNs, known as the LSTM model, was introduced by [69]. The LSTM model comprises a cell that can remember values on arbitrary time intervals and three gate structures that control information flow. The gate structures include input, forget, and output gates [2]. As a result, the LSTM classification method can better capture the connections among context feature words and use the forget gate structure to filter out useless information, improving the classifier's overall performance [2, 48].

Another variant of LSTM, Bi-LSTM, was introduced by [70] in 2005 and reads the input sequence in both forward and backward directions, allowing the network to capture context from both the past and future, which makes Bi-LSTM suitable for tasks that require a deep understanding of text [71]. GRU, invented by a team of researchers at Google led by Kyunghyun Cho in 2014 [72], is another type of RNN similar to LSTM but has a simpler architecture with fewer parameters. It uses gating mechanisms to selectively update or reset the cell state, making GRUs faster to train and computationally more efficient than LSTM models [73]. There are two gates: the reset gate and the update gate. The update gate determines how much of the previous hidden state is retained and how much new input is added. The reset gate determines how much of the last hidden state is forgotten to capture further information in the current input. During training, the GRU learns to adjust the weights associated with these gates to optimize the classification performance [71, 73].

Convolutional Neural Network

CNNs were initially proposed by [74] in 1998 for image classification by convolving filters that could extract features from pictures. Unlike RNNs, CNNs can apply convolutions defined by different kernels to multiple chunks of a sequence in parallel [2]. Although RNNs are adequate for NLP tasks that require understanding long-range dependencies, CNNs excel in detecting local and position-invariant patterns. These patterns may include crucial phrases expressing a specific sentiment or a topic [2]. As a result, CNNs have become one of the most widely used model architectures for text classification [48].

The operation of CNNs for text classification involves splicing the word vectors of the input text into a matrix and feeding it into a convolutional layer, which contains several filters with different dimensions [2]. Finally, the result of the convolutional layer goes through a pooling layer, and the pooling result is concatenated to obtain the final vector representation of the text, which predicts the category [2]. Although numerous variants of CNNs exist, such as the Deep Pyramid Convolutional Neural Network (DPCNN) by [75], this thesis focuses solely on the basic model.

Transformers

The Transformer model has acquired much traction in various domains and is currently the architecture of choice in NLP [48]. A group of Google researchers initially presented it in 2017 [76]. The sequential processing of text is one of the difficulties that RNNs must overcome [48]. While less sequential than RNNs, CNNs still struggle computationally to understand the associations between words in more extensive phrases. Transformers deal with this problem using self-attention, which simultaneously calculates an "attention score" for each word in a sentence or document [48]. Unlike RNNs or CNNs, this method dramatically increases parallelization by modeling the interaction between each word [48], which enables the pre-training of large models on vast amounts of data, resulting in a PLM. Furthermore, these PLMs can be fine-tuned to specific use cases by adjusting their internal weights [7]. There are two categories of PLMs: auto-regressive and auto-encoding [48].

One of the earliest auto-regressive PLMs is Generative Pre-trained Transformer (GPT), introduced by OpenAI in 2018 [77]. It generates text by predicting words sequentially from left to right (or right to left), where each word prediction depends on the previous prediction [48]. More powerful versions of GPT, namely GPT-2 and GPT-3, have since been released, but only GPT and GPT-2 are currently available as FOSS. On the other hand, BERT is a widely used auto-encoding PLM developed by Google researchers in 2018 [78]. Unlike GPT, which generates words based on previous predictions, BERT is trained using a masked language modeling task [7]. This task randomly masks some tokens in a text sequence and then independently recovers the masked tokens by conditioning on the encoding vectors obtained by a bidirectional Transformer [7].

Researchers made several improvements to BERT, resulting in various models, including Robustly Optimized BERT Pre-training Approach (RoBERTa), by Facebook's AI research in 2019 [79]. Trained using a much larger dataset, RoBERTa is more robust than its predecessor. Another approach to improving PLMs has been combining the strengths of auto-regressive and auto-encoding models, as seen in XLNet by [80]. This model integrates the auto-regressive practice of GPT and BERT's bi-directional context modeling. During pre-training, XLNet uses a permutation operation that allows context to include left and right tokens, resulting in a generalized order-aware auto-regressive language model [48].

In summary, RNNs compute sequentially and cannot be calculated in parallel, making it challenging for deeper models. CNNs can be parallelized and extract features from text vectors using a convolution kernel. Transformers extract features between words and address short-term memory issues by treating text as a fully linked graph with attention weights on the edges. Although efficient, transformers' attention technique requires much computation for lengthy sequences. Therefore, the appropriate model depends on a given NLP use case's requirements and limitations. Transformers are frequently preferred for text classification jobs due to their effective feature extraction capabilities and outperforming outcomes in many benchmark tests, such as [48]. However, although RNNs, CNNs, and Transformers all have various strengths and shortcomings, it is essential to carefully consider the specifics of the task at hand before selecting the most appropriate model.

3. Comparison Methodology

This section describes the methodology comparing various FOSS machine learning software against proprietary software offered by commercial providers. Therefore this section presents a dataset selected for analysis, which section 3.1 describes in detail. The dataset was first subjected to several preparatory steps, as outlined in section 3.2, to ensure the accuracy of the results. Next, section 3.3 delineates the execution of the FOSS models, incorporating Jupyter Notebooks, followed by a review of the commercial providers of machine learning models in section 3.4. Subsequently, section 3.5 discusses the metrics used for evaluating the performance and cost-effectiveness of the models. The best-performing FOSS model, as determined by the evaluation metrics, specifically the accuracy, was then pitted against the proprietary software to determine how it compares.

3.1. Description of the Dataset used for the Study

In the context of many businesses, the distinctions between different categories can be subtle, and the textual content may be intended for a specific, targeted audience rather than a general audience. For this reason, this thesis focuses on the Consumer Complaint Database[1] from the Consumer Financial Protection Bureau (CFPB). This U.S. government agency collects consumer complaints through formal, textual exchanges between companies, public institutions, and private individuals. This dataset represents a scenario similar to how private individuals typically communicate with businesses and therefore provides a relevant and applicable context for the study of document classification. The dataset got accessed on the 10th of January, 2023.

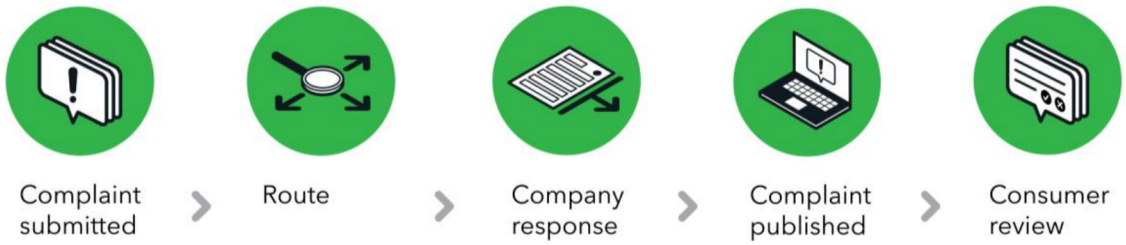


Figure 4: Consumer Complaints Process [81]

The dataset comprises 3,218,032 consumer complaints sent to the CFPB with all required details and forwarded to the financial institution in question, giving the latter a chance to respond. Finally, CFPB makes a subset of the complaint data publicly available

[1] https://www.consumerfinance.gov/data-research/consumer-complaints/

when fitting specific criteria (e.g., the absence of a lawsuit). Later the customer can file a review of the company's response [81]. Figure 4 shows the whole process of adding a record to the database. The dataset under consideration includes a range of information for each complaint, including *Date received, Product, Sub-product, Issue, Sub-issue, Consumer complaint narrative, Company public response, Company, State, ZIP code, Tags, Consumer consent provided, Submitted via, Date sent to the company, Company response to consumer, Timely response, Consumer disputed, and Complaint ID*. However, for this thesis, the focus will be solely on the textual content provided in the *Consumer complaint narrative* and the class provided by the *Product* column. The CFPB divided the complaints into 18 distinct categories initially. The distribution of complaints across these categories is shown below, and it is readily apparent that the data is significantly imbalanced.

Product	Complaints
Credit reporting, credit repair services, or other personal consumer reports	1,525,311
Debt collection	459,081
Mortgage	367,567
Credit card or prepaid card	170,426
Checking or savings account	146,542
Credit reporting	140,429
Credit card	89,190
Bank account or service	86,206
Student loan	72,251
Money transfer, virtual currency, or money service	49,369
Vehicle loan or lease	38,398
Consumer Loan	31,596
Payday loan, title loan, or personal loan	25,874
Payday loan	5,543
Money transfers	5,354
Prepaid card	3,819
Other financial service	1,058
Virtual currency	18
Overall	**3,218,032**

3.2. Data Preparation

This section focuses on data preparation, a crucial step in any data analysis project, as it has a tremendous impact on the performance of the classifiers, as previously explained in section 2.3. In this section, the dataset is cleaned and balanced in section 3.2.1, followed by preprocessing of the text using various techniques in section 3.2.2. The dataset preparation involves the removal of redundant, noisy, or irrelevant data to improve the dataset's quality. Balancing the dataset ensures that each class is nearly equally represented to avoid creating bias. In addition, preprocessing techniques such as tokenization, stop word removal, and punctuation removal are applied to the text data to enhance the effectiveness of the machine learning algorithms used in the analysis. Overall, data preparation is essential in ensuring that the dataset used in this study is suitable for analysis and that the results obtained from the research are reliable and accurate.

3.2.1. Dataset Preparation

This section will cover the problem of imbalanced data within the dataset and proposed solutions. Imbalanced data is a situation in which the quantity of samples from one class greatly outnumbers those from another. As further discussed in section 2.3.2, this might result in biased models that underperform for underrepresented classes. Data balancing addresses this issue since it seeks to redistribute the samples in the dataset so that the number of data points in each category is about equal. The problem of unbalanced data was reduced using this procedure, which enhanced the performance of the models. The accompanying Dataset Preparation Notebook[1] represents a comprehensive overview of the methodology employed to balance the dataset.

Since the dataset included several complaints without accompanying textual content, and the models classify complaints based on this content, all such cases of empty values were excluded. This exclusion resulted in the removal of 2,056,568 complaints from the dataset. Additionally, certain products were consolidated to streamline the classification process, which was inspired by [82]:

[1] `https://colab.research.google.com/drive/10vfHyaNdtTaYq7TEIpW1b_kTgN2coIip?usp=sharing`

- *"Credit card"* → *"Credit card or prepaid card"*
- *"Prepaid card"* → *"Credit card or prepaid card"*
- *"Money transfers"* → *"Money transfer, virtual currency, or money service"*
- *"Virtual currency"* → *"Money transfer, virtual currency, or money service"*
- *"Bank account or service"* → *"Bank, checking or savings account"*
- *"Checking or savings account"* → *"Bank, checking or savings account"*
- *"Credit Reporting"* → *"Credit reporting, credit repair services, or other personal consumer reports"*
- *"Payday loan"* → *"Payday loan, title loan, or personal loan"*

The category *"Other financial service"* has no specific or meaningful definition and was therefore eliminated from the dataset. The consolidation and the following removal resulted in a reduction from 18 initial classes to 10 remaining categories. In addition, after all redundant data entries were removed, a cap of 9,426 complaints per category was implemented to mitigate imbalanced data across categories. This number refers to the maximum number of complaints that any class contains, equal to the total number of complaints in the smallest category. As a result of this procedure, the following categories were retained:

Product	Complaints
Bank, checking or savings account	9,426
Credit card or prepaid card	9,426
Credit reporting, credit repair services, or other personal consumer reports	9,426
Debt collection	9,426
Vehicle loan or lease	9,426
Mortgage	9,426
Payday loan, title loan, or personal loan	9,426
Money transfer, virtual currency, or money service	9,426
Student loan	9,426
Consumer Loan	9,426
Overall	**94,260**

Finally, the dataset was divided into two sets for training and testing. The training set comprises 85% of the total data, or 80,121 documents, while the test set comprises the remaining 15%, or 14,139 documents. This approach was implemented to guarantee that each model receives the same training data and is subsequently evaluated on the same set of documents. This step created the first dataset, as PLMs such as BERT and RoBERTa do not require textual preprocessing but rather extract information from capitalization, punctuation, and other similar features [83, 84].

3.2.2. Textual Preprocessing

As previously outlined in Section 2.4, preprocessing the data's textual content is paramount as it can significantly influence the model's performance. To provide a clear understanding of how the preprocessing affects the utilized dataset, an example (Complaint ID: 5937822) has been provided to demonstrate the outcomes of the preprocessing. Furthermore, the Text Preprocessing Notebook[1] gives a detailed overview of all the steps and intermediate results for each step.

Raw Sample:
"I initially obtained a loan in XX/XX/XXXX for a car lease for a XXXX XXXX XXXX, I obtained a loan from XXXX XXXX XXXX XXXX XXXX, as of XXXX it was changed to XXXX. I am in Predatory loan and need help out ; I am XXXX upside down in this loan from XXXX they were going to report me to the credit agency for $27.00 yes XXXX dollars. When you ask for extension, they put so much interest on the deferred payment plan. I don't think I will ever be done paying for this loan, Please Help."

As several studies [83, 84] indicate that PLMs, such as BERT, perform better when no preprocessing is applied, the first dataset consists of samples that look similar to the raw sample above. In addition to the first dataset created, which did not involve any preprocessing, one additional dataset was generated using different text processing strategies, involving the removal of punctuation, the removal of the placeholder character "X" used to replace confidential user information, the removal of digits, the removal of excessive whitespace, the removal of stopwords, and the removal of non-ASCII characters. The preprocessed example is shown below:

[1] `https://colab.research.google.com/drive/1in0CLK2aOCuzsYlCLqyAyg28B3YNE6YC?usp=share_link`

Preprocessed Sample:

"initially obtained loan car lease obtained loan changed predatory loan need help upside loan going report credit agency yes dollars ask extension put much interest deferred payment plan dont think ever done paying loan please help"

The whole preprocessing reduced the dimensionality of the input features by about 31%, from an initial 34,016 unique tokens to 23,486 unique tokens, which will probably improve the performance of the traditional classifiers and the neural networks.

3.3. Free Open Source Models Implementation

The document classification models were implemented using Python notebooks in Google Colab, which provided an efficient and convenient connection to Google Drive for loading and saving datasets, dumping the models, and comparing results. Google Colab also enabled linking the notebooks to this thesis for a comprehensive overview of the model implementation. This thesis explored three groups of models: traditional machine learning methods such as NBCs, KNN classifiers, decision tree classifiers, logistic regression classifiers, and SVMs; neural network-based models including LSTM, GRU, and CNN; and transformer-based models such as BERT, RoBERTa, XLNet, and GPT-2. Each model was trained on the same labeled dataset discussed in section 3.1 after cleaning and balancing as described in section 3.2.1. In addition, the transformer-based models worked with raw text samples, while the other models used preprocessed text samples, as discussed in section 3.2.2. While some common hyperparameters were set for each model, this study focuses on providing a general overview of each model rather than the most optimized implementation, meaning an adequate hyperparameter optimization could potentially improve performance.

The traditional models were implemented using Scikit-learn [85], a popular open-source machine-learning library for Python that provides a range of supervised and unsupervised learning algorithms. It was chosen for its user-friendliness, efficiency, and consistent Application Programming Interface (API), making switching between algorithms and models easy. The models were all trained with the BOW, TF-IDF, and LSA feature extraction methods, while the best prediction performance was chosen. The Traditional Methods Notebook[1] presents a comprehensive overview of these models' implementation. The neu-

[1] `https://colab.research.google.com/drive/1Z9J8mcEqvDKzNNWkYmB7fv5QCFdQ9GHI?usp=share_link`

ral network-based models were implemented using Keras [86], a high-level open-source neural network library for Python that enables fast experimentation with deep learning models. It is built on top of lower-level deep learning frameworks, such as TensorFlow and Theano, which provide the computational backend for Keras. This thesis used the Keras standard layer for word embedding, i.e., the word embeddings were trained from scratch. The Neural Network Based Notebook[1] presents the whole implementation process of these models. Finally, the transformer-based models were implemented using Hugging Face Transformers [67], an open-source library built on top of PyTorch and TensorFlow that provides state-of-the-art NLP capabilities and offers a range of pre-trained models and tools that make it easy to develop and fine-tune models. The Transformer Based Notebook[2] presents the implementation process.

3.4. Overview of Proprietary Software

Several software providers offer tools to help businesses and researchers in document classification. This thesis will focus on several popular software providers and their products: Microsoft Azure, Amazon Comprehend, OpenAI GPT-3, Levity AI, Planet AI, and Vertex AI.

- **Microsoft Azure** [87] is a cloud-based platform that offers a range of AI and machine learning services, including Azure Cognitive Services for document classification. The service offers various features like text analytics, language detection, and sentiment analysis. Azure's Cognitive Services for text classification uses pre-built machine learning models and algorithms, making it a user-friendly option for those without extensive machine learning expertise. In addition, Azure offers a free trial for its "Cognitive Services", allowing users to experiment with the platform before committing to a paid subscription.

- **Amazon Comprehend** [88] is a machine learning-based tool that can analyze text data to extract relevant information and provides various services, including custom document classification, which this study utilized. Additionally, Amazon Comprehend supports multiple languages, including English, Spanish, French, German, Italian, and Portuguese.

[1] `https://colab.research.google.com/drive/14aEItDN1VwwYpiMDuqxLhkgEC-sFLKbr?usp=share_link`
[2] `https://colab.research.google.com/drive/1ovAauNsgHLOY3SpNju8yRkpE8AlxE3vb?usp=share_link`

- **OpenAI's GPT-3** [89] is a natural language processing tool using deep learning algorithms to generate text. The tool is handy for text generation tasks, such as language translation, chat-bots, and content creation. Through its recently released product "ChatGPT", OpenAI has gained massive attention. However, GPT-3 does not specialize in document classification but in text generation [90], so it may not be the best choice for those explicitly seeking text classification capabilities. OpenAI offers four models: "Ada", "Babbage", "Curie", and "Davinci" (from weakest/cheapest to strongest/most costly). For this thesis, "Baggage" was chosen, due to the recommendation in OpenAI's documentation.

- **Levity AI** [91] is a machine learning-based document classification tool that offers a range of features, including text categorization, sentiment analysis, and document clustering. Levity's designed its platform to be very user-friendly, with an intuitive interface and drag-and-drop tools for data preparation. After arranging it with its sales team, Levity AI offered a free trial of its platform.

- **Planet AI** [92] is another machine learning-based tool for document classification that uses deep learning algorithms to analyze text data. The platform offers various features like entity recognition, sentiment analysis, and topic modeling. Planet AI also provided a free trial of its platform after arranging it with its sales team. However, it is to mention that Planet AI's primary focus is on visual-language modeling tasks, and the dataset used for the study only consists of text, which is why the strengths of the company are not being tested here.

- **Google Vertex AI** [93] is a cloud-based AI platform offering a range of machine learning and deep learning tools, including document classification. Vertex AI allows users to train and deploy models quickly and easily with built-in data preparation, training, and evaluation tools. In addition, Google offers a free trial for Vertex AI, allowing users to test the platform before committing to a paid subscription.

All service providers underwent training through their respective websites; however, they all provided an API option, allowing for convenient testing under identical conditions in Google Colab. The implementation details for each model are available in the Proprietary Software Notebook[1].

[1] `https://colab.research.google.com/drive/1GpZjxm1MUHo7BYIgfkMdRAbnF0FoNi4L?usp=share_link`

In summary, the tools and software providers for document classification offer a range of options for businesses and organizations looking to analyze text data. Some providers offer free trials or pay-as-you-go pricing models, while others require substantial commitments. The best choice for a particular organization will depend on its specific needs and requirements, such as the size of the data set, the complexity of the analysis, and the level of machine learning expertise available within the organization.

3.5. Evaluation Metrics and Methodology for Comparison

This section explores the evaluation metrics and methodology used to compare the performance of the FOSS models presented in section 3.3 and the proprietary software presented in section 3.4. Evaluation is a crucial aspect of any machine learning project, as it provides insight into how well the models are performing and whether they meet the desired performance criteria. This section begins by discussing the basic concepts of performance evaluation metrics, such as the confusion matrix and their components, and some metrics used for comparisons like accuracy and F1-Score in section 3.5.1. Then, this thesis utilizes these metrics to measure the prediction performance of the models on test data and provide insights into the strengths and weaknesses of each model. Additionally, this thesis evaluates the computational efficiency of the models. Section 3.5.2 discusses the metrics and calculations used to compare the models cost-wise. Cost evaluation metrics are essential for identifying the most cost-effective model regarding time and resources. This section discusses the importance of these metrics in selecting the most suitable model for the document classification task at hand. Overall, this section highlights the importance of evaluation metrics and methodology for comparing the performance of deep learning models developed for document classification tasks. The discussion of both performance-wise and cost-wise evaluation methods provides a comprehensive framework for evaluating and comparing the models, allowing businesses to make informed decisions about the most suitable model for the task at hand.

3.5.1. Performance-wise

In assessing multi-class machine learning models, various metrics impart information on the model's effectiveness. The following section offers a succinct overview of the critical metrics employed. The discussion refers to [94] and [95]; the reader is encouraged to consult the original papers for a more comprehensive understanding.

Initially, this section establishes an understanding of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). In binary classification, the computation of these metrics is straightforward:

- **TP:** The number of positive instances the model correctly classified as positive
- **TN:** The number of negative instances the model correctly classified as negative
- **FP:** The number of negative instances the model incorrectly classified as positive
- **FN:** The number of positive instances the model incorrectly classified as negative.

A confusion matrix, a cross table that records the number of occurrences between two raters, visualizes these metrics, as shown in Figure 5.

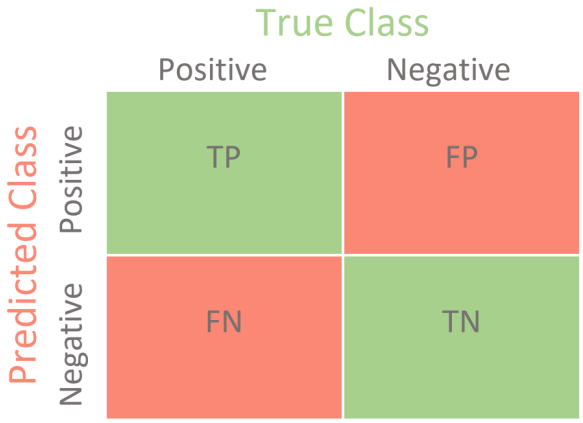


Figure 5: Binary Confusion Matrix [96]

In multi-class classification, these metrics are extended by calculating each class's TP, TN, FP, and FN individually and then aggregating these values to get a single measure of the classifier's prediction performance. One standard method for doing so is to consider each class as a binary classification problem, with the instances of that class being positive and the instances of all other classes being negative. This "one-vs-all" approach allows calculating each class's TP, TN, FP, and FN and then averaging these values across all classes to measure the classifier's prediction performance. Based on these confusion matrix metrics, a wide range of prediction performance measures can be derived, which provide a comprehensive evaluation of the prediction performance of a classifier. This thesis uses several of these measures to evaluate the models. A brief overview of the binary measures is shown below:

- **Accuracy:** Accuracy is one of the most popular metrics in multi-class classification and describes the probability that the model's prediction is correct:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

- **Precision:** Precision tells how much the model can be trusted when it predicts an instance as positive:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

- **Recall:** Recall measures the model's predictive accuracy for the positive class: intuitively, it measures the ability of the model to find all the positive instances in the dataset:

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

- **F1-Score:** The F1-Score can be considered a weighted average of precision and recall. The harmonic mean can be used to determine the optimal trade-off to balance the relative contributions of precision and recall to the F1 score:

$$F1\text{-}Score = 2 * \left(\frac{Precision * Recall}{Precision + Recall} \right) \tag{13}$$

Regarding multi-class classification, there are two different methods for calculating the overall measure for the classifier. The "macro-average" gives all classes the same weight, disregarding whether a class is densely or sparsely populated and instead using the average measurement score for each class. However, the "micro-average" considers the confusion matrix variables together, giving each sample the same weight rather than each class. As the dataset utilized for this study is reasonably balanced, this thesis solely presents the macro-average scores. Nevertheless, both micro- and macro-average scores have been reported in the corresponding FOSS Results Notebook[1] to provide a more comprehensive picture of the classifier's performance. Given that the F1-Score is the harmonic weight between precision and recall, this thesis solely displays the F1-Score within the evaluation table. In addition to the previously specified prediction performance metrics, the duration of model training and inference will be tracked in milliseconds and averaged for each document to gain insights into the computational efficiency.

[1] `https://colab.research.google.com/drive/14QkxaD_vDOfTNKEUSZMt6yyJF41C23Ly?usp=sharing`

3.5.2. Cost-wise

This section provides a comprehensive outline of the comparison methodology used to evaluate the cost-effectiveness of various document classification products. The comparison process entails assessing three principal costs associated with the products, specifically training costs, hosting costs, and inference costs, which constitute the total pricing of the products in most cases. This thesis simulates a 30-day period, during which the classification model will be trained on the first day, hosted for the entire duration, and inference costs computed for varying document volumes. In addition, the hosting costs for the whole period will be added to the training costs at the outset to provide a clearer view of the costs. The primary objective of this comparison is to identify the most cost-effective product for document classification, considering varying document volumes for a month. The pricing for training, hosting, and inference was obtained from the respective providers' websites.

Furthermore, following the methodology introduced in Section 3, this thesis will evaluate the proprietary software against the best-performing FOSS model. However, this comparison presents some challenges as the pricing of the FOSS model is contingent on the implementation method. Therefore, this study focuses on a widely used variant, Amazon SageMaker. Amazon SageMaker is a fully managed machine learning service provided by Amazon Web Services (AWS) that facilitates the development, training, and deployment of machine learning models on a large scale. The service includes various pre-built algorithms and frameworks, making it a comprehensive platform for managing the entire machine-learning workflow. The FOSS model, previously trained on Google Colab, will be uploaded and integrated into the workflow illustrated in Figure 6. The workflow encompasses the cost of storing the FOSS model artifacts in an S3 bucket, generating an Elastic Container Registry (ECR) container image, and hosting the endpoint for the model. Appendix B and Section 4.3 present a detailed analysis of all costs based on the AWS Pricing Calculator.

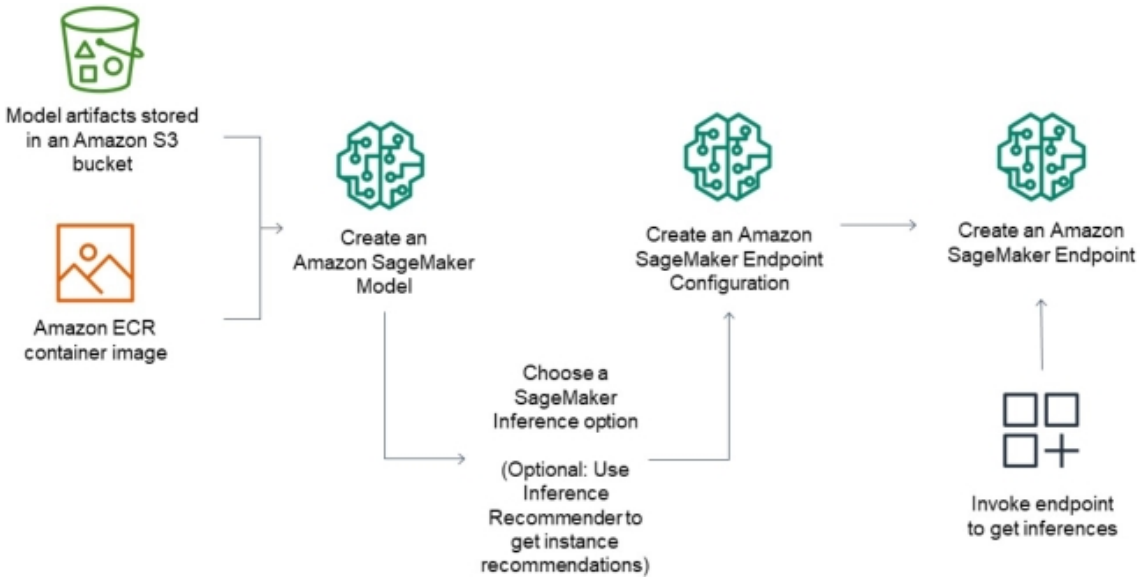


Figure 6: Amazon SageMaker Workflow [97]

It is essential to acknowledge that a cost comparison between different commercial providers for document classification systems may not be entirely equitable, as this comparison does not consider additional factors beyond the base cost of the service, such as pricing for ancillary services like cloud storage, the level of effort required by the user to train and adjust the model for their specific needs, and any API connection points that may be included in the pricing structure. These additional considerations may play a significant role in the total cost of ownership and the overall value proposition of a given provider. The same goes for the implementation of the FOSS model, which does not include additional charges for factors beyond the base cost of the service, such as local/cloud data storage, model maintenance, or human resource cost.

4. Results

This section presents the experiments' results on different document classification models and compares the performance of various FOSS models in section 4.1. This section discusses the prediction performance of the models developed using open-source software, including their accuracy and F1-Score. Additionally, this section provides an overview of their computational efficiency. Section 4.2 selects the best-performing FOSS model, in terms of accuracy and F1-score, from section 4.1 as the baseline for comparison with commercial software providers and evaluates the performance of proprietary software providers. The results are then compared to the FOSS model to identify any significant differences in performance. Section 4.3 compares the costs of the commercial software providers and the selected FOSS model. This section examines various cost factors, such as training, hosting, and inference, to provide a comprehensive overview of the cost differences between the models. Finally, section 4.4 analyzes and discusses the results obtained from the experiments by providing insights into the strengths and weaknesses of each model and discussing the implications of the results obtained. Overall, this section presents a comprehensive analysis of the performance and cost comparison of different document classification models, providing valuable insights into the best models for document classification. The results obtained in this study may be specific to the dataset and use case considered in this study. Therefore, different datasets and use cases may produce different results, and it is essential to evaluate the models on the specific use case to determine the most suitable model.

4.1. Performance of Free Open Source Software

This section presents the performance metrics of various FOSS models used in the document classification task, categorized as traditional machine learning, neural network, and transformer-based models. Table 2 displays each model's accuracy and F1-Score, along with the average training and inference time in milliseconds per document. In addition, Appendix A provides each model's confusion matrix for better visualization of the scores for each class, and the FOSS Results Notebook[1] delivers a more comprehensive overview of all the scores.

[1] `https://colab.research.google.com/drive/14QkxaD_vDOfTNKEUSZMt6yyJF41C23Ly?usp=sharing`
[1] Google Colab's premium GPU was being used
[2] Google Colab's standard GPU was being used

Model	Accuracy	F1-Score	Training Time	Inference Time
NBC	74.06	73.61	**0.47**	12.25
KNN	44.76	44.85	0.48	725.39
Decision Tree	61.84	61.77	11.41	**1.55**
Logistic Regression	76.49	76.25	38.06	17.99
SVM	76.69	76.57	316.13	106.52
LSTM	73.92	73.75	257.55	115.08
Bi-LSTM	72.96	72.82	264.59	127.37
GRU	74.67	74.09	68.54	71.55
CNN	71.03	70.97	4.16	49.24
BERT	79.57	79.41	78.65[1]	20.49[2]
RoBERTa	**81.29**	**81.25**	316.02[2]	22.15[2]
XLNet	80.20	80.20	168.97[1]	36.95[2]
GPT-2	78.26	78.15	217.56[1]	29.83[2]

Table 2: FOSS Results

The metrics for the traditional machine learning models show that the logistic regression and SVM models perform the best in accuracy and F1-Score. In contrast, KNN and decision tree models show the worst prediction performance. The unsatisfying results of the decision tree classifier might be due to the high dimensionality of the data and the significant number of relevant features. Additionally, the long inference time of KNN is due to its known computational intensity, mainly when dealing with large datasets, as explained in section 2.6.1. Finally, the poor prediction performance might be due to the wrong choice of the "k" nearest neighbors.

Among the neural network-based models, the best-performing models in terms of accuracy and F1-Score are GRU and LSTM; their prediction performance is still slightly lower than the best traditional machine learning models. The Bi-LSTM model also performs well, while the CNN model performs the worst among the neural network-based models, potentially because CNNs are mainly used for image and video processing tasks, as explained in section 2.6.2, where data has a grid-like structure, whereas RNNs (LSTM, Bi-LSTM, and GRU) are used for processing sequential data, such as natural language text and speech.

Finally, transformer-based models perform better than traditional machine learning and neural network-based models in all prediction performance metrics. RoBERTa has the best prediction performance with an accuracy of 81.29% and a F1-Score of 81.25%. The other transformer-based models, BERT and XLNet, also perform very well. The slightly lower

prediction performance of the GPT-2 model could be due to its primary design for tasks such as language translation, language generation, and language understanding rather than document classification. Notably, the transformer-based models had a very high training and inference time, which was reduced using a Graphics Processing Unit (GPU). Nevertheless, as RoBERTa had the best prediction performance among all the models for this task, it will be the comparison model for evaluating the proprietary software.

4.2. Performance of Proprietary Software

This section presents the prediction performance metrics of the proprietary software, presented in section 3.4, in the same way as section 4.1. Table 3 displays each model's accuracy and F1-Score, together with the average training and inference time in milliseconds per document. RoBERTa, the best-performing FOSS model, can be found as a baseline at the bottom of the table. The Proprietary Results Notebook[1] presents a more comprehensive overview of the performance.

Model	Accuracy	F1-Score	Training Time	Inference Time
Microsoft Azure	79.13	79.03	573.97	5439.81
Amazon Comprehend	78.04	78.01	260.68	14920.47
OpenAI GPT-3	**81.27**	**81.18**	201.85	96.82
Levity AI	80.29	80.27	261.35	1491.67
Planet AI	79.04	79.01	**132.21**	**27.24**
Google Vertex AI	78.12	77.95	217.92	350.91
RoBERTa	*81.29*	*81.25*	*316.02*	*22.15*[2]

Table 3: Proprietary Results

Table 3 provides a comparative analysis of different commercial providers and the open-source baseline model, RoBERTa. The models' prediction performance is evaluated on accuracy and F1-Score. Additionally, the training and inference times of the models are also recorded. The results show that GPT-3 outperforms all other models in all prediction performance metrics, with an accuracy of 81.27% and a F1-Score of 81.18%. On the other hand, Amazon Comprehend scored the lowest in these metrics, with an accuracy of 78.04% and a F1-Score of 78.01%. Notably, all models show similar prediction performance, with the highest difference in accuracy being 3.23%.

[1] `https://colab.research.google.com/drive/1TJwTD6GXUY4R1gfl0fBSNDEzFQcyIi2U?usp=share_link`
[2] inference did not include an API

However, there are significant variations in training and inference times. Planet AI demonstrates the shortest training time of 132.21ms on average per document and the shortest inference time of 27.24ms on average per document among all proprietary software. In contrast, Azure takes the longest training time of 573.97ms on average per document, approximately four times the training time of Planet AI. Aside from that, Amazon Comprehend has the longest inference time of 14920.47ms on average per document, which is approximately 548 times the inference time of Planet AI. Although GPT-3 outperforms all other models in prediction performance metrics, it falls short of the open-source baseline model, RoBERTa, which achieves an accuracy of 81.29% and a F1-Score of 81.25%. Despite its lower prediction performance metrics, Planet AI outperforms RoBERTa regarding training but not inference time. However, it should be noted that the inference of RoBERTa was made in-house and on a GPU, whereas Planet AI's inference was performed through an API and a single-core Central Processing Unit (CPU), which slowed the process.

The findings shows that GPT-3 performs well in all performance metrics and achieves similar results as the FOSS baseline model, RoBERTa. Planet AI has the shortest training and inference times among all proprietary models, whereas Microsoft Azure and Amazon Comprehend demonstrate the longest training and inference times, respectively. If the inference time of RoBERTa was performed through an API and on a CPU, Planet AI could also outperform it regarding inference time. It should be noted that Levity AI, which is relatively less well-known in this comparison, achieved a highly competitive prediction performance compared to industry giants such as Google's Vertex AI, Microsoft's Azure, and Amazon Comprehend.

4.3. Cost Comparison

This section compares the various products offered by commercial providers against each other cost-wise. To do so, a 30-day time period was simulated. A new classification model was trained at the start of the period and hosted throughout the entire period. The costs were factorized for documents that needed to be classified within this period. For the whole period, there are three main cost components: training, hosting, and inference costs per document. The formulas for calculating the training, hosting, and interfering costs were accessed on the service providers' corresponding websites, found in references [87] to [93], on the 15$^{\text{th}}$ of February, 2023, and were applied to the dataset.

This section introduces several variables to enhance the clarity of the cost formulas shown in Table 4: h represents an hour, m represents a month, b_t signifies the ratio of 1000-character text blocks rounded up to the nearest block, which is equal to 1.75, b_h signifies the ratio of 100-character text blocks rounded up to the nearest block, which is equal to 12.74, and t denotes the average number of tokens on each document, which is equal to 276.63. For clarification, a document containing 250 words and 1250 characters would equal two 1000-character text blocks, 13 100-character text blocks, and roughly 330 tokens. It is to mention, that GPT-3 counted four times more tokens (966) for the training data, so the billing is significantly higher. As some providers settle the training costs through the training time, the training time was measured, as shown in Table 3, and the costs were calculated. Finally, as Amazon bills per second of endpoint hosting for real-time inference, it was calculated how long it takes to classify one document based on the average number of characters per document. Amazon uses "Inference Units", which can process one 100-character text block per second. This thesis utilized one inference unit.

$$\frac{s}{document} = 1 \ \frac{\text{100-character-block}}{s} \cdot b_h = 12.74 \ \frac{s}{document} \tag{14}$$

$$\frac{costs}{document} = 0.0005 \ \frac{\$}{s} \cdot 12.74 \ \frac{s}{document} = 0.00637 \ \frac{\$}{document} \tag{15}$$

As stated in section 3.5.2, RoBERTa was hosted via AWS SageMaker. AWS SageMaker includes a variety of costs which are listed below:

- **Model artifacts:** After training the RoBERTa model on Google Colab without any costs, the model artifacts must be uploaded into an Amazon S3 bucket to make it available for Amazon SageMaker. As the model is around 300MB in size and needs to be stored for 30 days with up to 100,000 requests, the costs calculate as follows:

$$S3 \ costs = 0.3GB \cdot \$0.023/GB + 100,000 \cdot \$0.000005/request = \$0.51 \tag{16}$$

- **ECR container image** The model needs to be contained in an ECR container image to make it available for Amazon SageMaker. The costs for this calculate as follows:

$$ECR\ costs = 0.3GB \cdot \$0.1 = \$0.03 \tag{17}$$

  Adding these costs to storing the model artifacts results in training costs of $0.54.

- **Endpoint hosting** For hosting the model, a "ml.m5.large" instance was chosen. The costs for hosting this instance 24 hours a day for 30 days with one model and one endpoint calculate as follows:

$$hosting\ costs = 24h \cdot 30d \cdot \$0.115 = \$82.80 \tag{18}$$

- **Inference** With the assumption that one document x is 1MB in size, which is already very generous, the inference costs would calculate as follows:

$$inference\ costs = x \cdot 1MB \cdot \$0.000015625/MB \tag{19}$$

A more comprehensive overview of the cost composition for RoBERTa can be found in Appendix B. **Levity AI** employs a monthly subscription plan that encompasses training, hosting, and processing a specified number of documents, hence the absence of this company from the pricing table. Furthermore, the inference numbers of **Planet AI** typically exceed those of the simulation by a considerable margin, and owing to the complexity of its internal cost structure, it was not feasible to establish a comprehensive pricing scheme for the 30-day simulation, resulting in the exclusion of the company from the cost analysis. The costs for every other product offered by commercial providers and additionally for RoBERTa can be found in Table 4.

Service	**Training**	**Hosting**	**Inference**
Microsoft Azure	$\$3/h$	$\$0.5/m$	$\$0.005/b_t$
Amazon Comprehend	$\$3/h$	$\$0.5/m$	$\$0.00637/document$
GPT-3 (Babbage)	$\$0.0000006/t$	\$0	$\$0.0000024/t$
Vertex AI	$\$3.30/h$	$\$0.05/h$	$\$0.005/b_t$
RoBERTa	\$0.54	\$82.80	$\$0.000015625/document$

Table 4: Costs Formulas

A line graph has been constructed in Figure 7 to facilitate the comparison of costs between different approaches. The chart displays the training costs accumulated with the hosting costs of each model at the beginning of the chart, as well as the costs incurred from processing different numbers of documents.

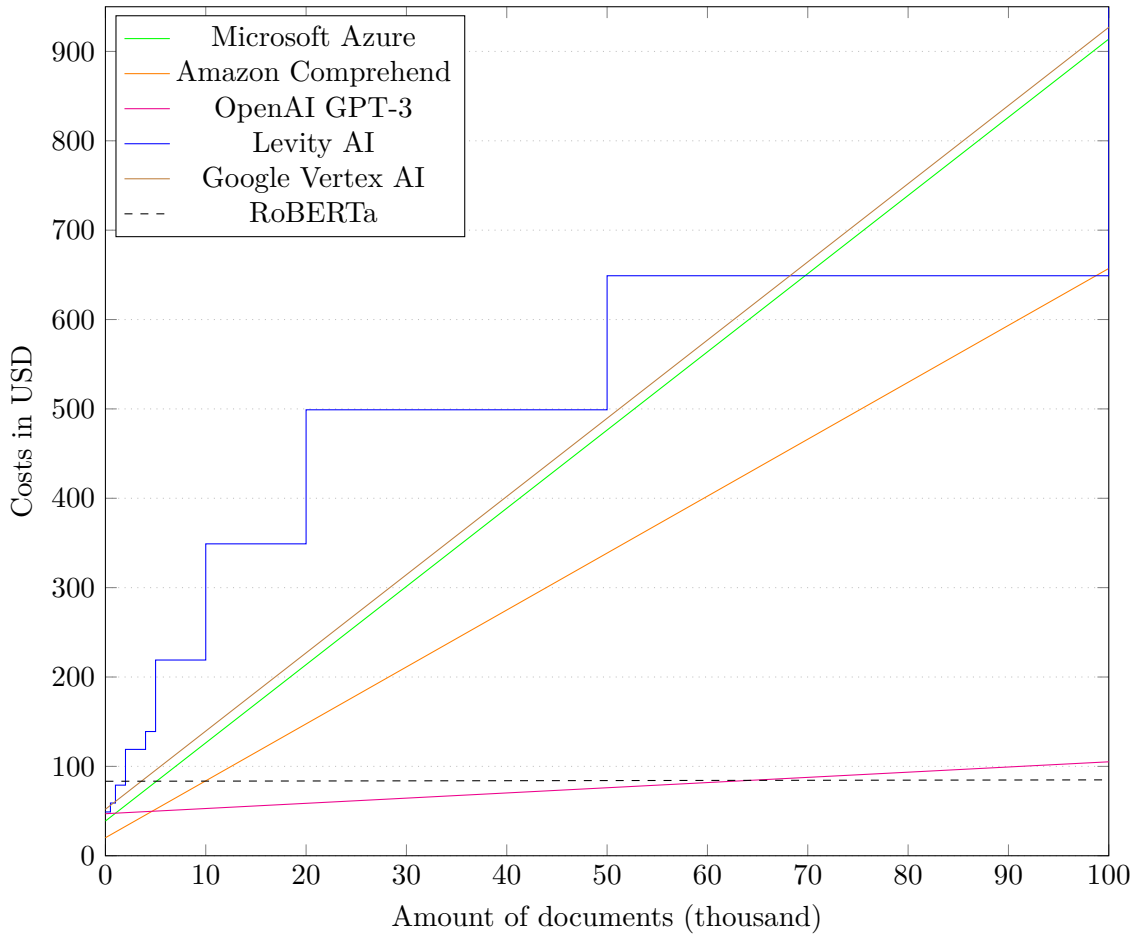


Figure 7: Cost Comparison

According to the findings, there are considerable differences in the total expenses between various suppliers. For example, the combined training and hosting expenses, further referred to as *upfront costs*, for Microsoft Azure are very moderate. However, its inference costs are comparatively high, making its overall prices the second-highest at 100,000 documents. The most expensive supplier is Google Vertex AI, which charges the same amount for inference as Microsoft Azure but has higher upfront costs. The second-lowest costs for practically all document volumes are achieved by Amazon Comprehend, which begins with the lowest upfront costs and retains comparable low inference costs. Levity AI initially appears to price comparatively higher for smaller document volumes, but it ultimately

manages to overcome the costs of Microsoft Azure, Amazon Comprehend, and Google Vertex AI. Furthermore, despite having the highest upfront costs, the GPT-3 model from OpenAI is the most affordable among all proprietary models for practically all document quantities. Importantly, OpenAI's GPT-3 expenses at 100,000 documents are almost a ninth of Amazon Comprehend's. Finally, although RoBERTa has the highest upfront costs due to hosting the "ml.m5.large" instance, its negligible inference costs make it the cheapest option among all providers at higher document volumes. Again, however, GPT-3's costs are comparable.

4.4. Analysis and Discussion

Starting with the prediction performance of the FOSS, it can be observed that some models perform better than others in terms of accuracy and F1-Score. RoBERTa has the highest accuracy and F1-Score with 81.29% and 81.25%, respectively, while KNN has the lowest accuracy and F1-Score with 44.76% and 44.85%, respectively. In terms of training time, there are significant differences. SVM has the longest training time with 316.13ms on average per document, followed by RoBERTa and Bi-LSTM. On the other hand, the NBC has the shortest training time with 0.47ms on average per document, followed by KNN. Inference time also varies widely across the models. KNN has the longest inference time with 725.39ms on average per document, followed by Bi-LSTM and LSTM. The decision tree classifier has the shortest inference time, with 1.55ms on average per document, followed by NBC. Notably, the training and inference process of the PLMs were accelerated with a GPU, which made them much faster.

However, it is essential to note that different models may have different strengths and weaknesses. By analyzing these and applying techniques such as dimensionality reduction, which might drastically improve KNN's and decision tree's prediction performance, or different word embedding techniques, which might improve the prediction performance of CNN and RNNs, the approaches can be optimized. Overall, the choice of the best model depends on the specific requirements of the problem, including the trade-off between accuracy, training time, and inference time. For example, if high accuracy is the most crucial factor, as it was for this thesis, RoBERTa may be the best choice, even though it has a long training time. On the other hand, if fast inference time is more important, decision tree or NBC may be a better choice, even though they may have lower accuracy than other models.

By comparing the performance of the proprietary software, it can be observed that the products perform similarly in prediction performance, with GPT-3 having the highest accuracy and F1-Score with 81.27% and 81.18%, respectively. However, there are significant differences between the models regarding time measurements. Planet AI performs best with 132.21ms training time on average per document and 27.24ms inference time on average per document. Still, no proprietary software achieved better prediction performance than the base comparison model RoBERTa, although GPT-3 came quite close. However, time-wise, Planet AI achieved a better training time and probably better inference time under the same conditions. Finally, looking at the costs for the products, it is observable that GPT-3 also has the lowest cost for inferring with 100,000 documents among the proprietary software, followed by Amazon Comprehend, Levity AI, Microsoft Azure, and Google Vertex AI. In contrast, RoBERTa's implementation via AWS SageMaker is still the cheapest option. However, it is to mention that several providers offer different options, such as GPT-3 with models that are slower and more accurate but therefore more costly, or Amazon Comprehend, which offers a scalable number of "Inference Units" that accelerate inference but also increase pricing. Overall, the choice of the best service depends on the specific requirements of the problem, including the trade-off between accuracy, training time, and inference time, as well as the cost of the services and other external characteristics, such as availability, user-friendliness, scalability, data encryption, integration options or other services.

Finally, it is to mention that as the dataset used for the study is publicly available, it can not be ruled out that one or more of the PLMs used this specific dataset, or part of it, in their pre-training data, which would, of course, falsify the results. The same goes for some commercial providers, which might also use PLMs. Moreover, as different models have different strengths and weaknesses, the results of using a different dataset may vary significantly.

5. Conclusion

The objective of this thesis was to compare the effectiveness of two types of software in the field of document classification: FOSS and proprietary software. The thesis provided a comprehensive explanation of the fundamental concept of the document classification workflow, which included a detailed overview of each step in the process. To test the various approaches, a dataset from the CFPB was introduced, and techniques for data balancing and text preprocessing were discussed. The implementation of the FOSS models, an overview of the proprietary software, and the evaluation metrics used for comparison were also explained. Finally, the study thoroughly examined relevant literature and conducted experiments to identify each approach's key strengths and limitations. Section 5.1 presents the study's main findings, highlighting the most significant outcomes and insights gained. Additionally, in section 5.2, areas for future research are proposed, discussing topics and avenues that warrant further exploration and refinement.

5.1. Summary of the Main Findings

First, several FOSS models were initially researched, including traditional statistic-based methods such as NBCs, KNN algorithms, decision tree classifiers, logistic regression, and SVMs. Next, deep learning approaches, including LSTMs, Bi-LSTMs, GRUs, CNNs, and PLMs such as BERT, RoBERTa, XLNet, and GPT-2, were examined. This comparison was achieved by training each model on the same training dataset and inferring on the same test dataset. The best performing FOSS model in terms of several evaluation metrics, primary accuracy and F1-Score, was then compared against proprietary software offered by commercial providers such as Microsoft Azure, Amazon Comprehend, OpenAI's GPT-3, Levity AI, Planet AI, and Google's Vertex AI. This comparison included evaluating overall prediction performance, a time-wise comparison, and a comprehensive cost-wise comparison.

Among the FOSS models, this thesis found that the PLM RoBERTa performed best in prediction performance with an accuracy of 81.29% and a F1-Score of 81.25%, while in contrast, having a very high training and inference time. Still, as this thesis focused primarily on the accuracy to choose the best-performing model, RoBERTa was set as the baseline model for a comparison with the proprietary software offered by commercial providers. All of the proprietary products achieved quite good results, with GPT-3 having

the highest accuracy and F1-Score with 81.27% and 81.18%, respectively, which is still slightly lower than the prediction performance of RoBERTa. Planet AI had the lowest training and inference time among all commercial solutions and a lower training time than RoBERTa. By simulating a 30-day period and calculating the costs for classifying a varying amount of documents, ranging from 0 to 100,000, within that period, the commercial providers were compared against each other. To include RoBERTa in the comparison, the model was hosted via AWS SageMaker for this period. By comparing the models cost-wise, this thesis found that GPT-3 had the lowest costs for nearly all documents by far. Compared to RoBERTa, the costs are very similar to that of GPT-3, while GPT-3 managed to have lower costs at fewer documents but higher costs at a higher amount of documents.

It is important to note that the performance depends on the specific use case and may vary significantly for different scenarios. Additionally, the FOSS models were not optimally hyper-tuned, which could improve their performance. Furthermore, it cannot be ruled out that some of the PLMs used the dataset, or part of it, in their pre-training, which would falsify the results. The same applies to commercial providers, which may also use PLMs. Overall, the best solution depends on the particular needs of the problem, the trade-off between accuracy, training time, inference time, the cost of the services, and other external factors like availability, user-friendliness, scalability, data encryption, integration options, or other services.

5.2. Suggestions for Further Research

The present study provides several suggestions for future research and work in document classification. Firstly, to enhance the significance of the findings, investigating methods for optimizing the model's implementation to achieve the best possible results is crucial. Specifically, adjusting the hyperparameters is necessary to improve the effectiveness of the models. Secondly, exploring the interpretability of the models is advised to understand how they arrive at their predictions and identify any potential biases in the data or models. This exploration will increase the transparency and trustworthiness of the models. Finally, a similar test using a different dataset is recommended to evaluate the models and products' performance across various use cases. These additional investigations can provide further insights into the applicability of FOSS models and proprietary software.

To broaden the scope of application, a recommendation is to increase the number of FOSS models and commercial providers. Furthermore, since GPT-3 outperformed all proprietary software, a similar study can be conducted to compare GPT-4's performance with other models, as it will soon be released. Another area of interest is investigating how to adapt the models to handle multilingual documents, which can significantly broaden the scope of the classification task. Furthermore, besides Planet AI, most of the proposed software for document classification primarily focuses on textual content while neglecting the style and layout of the documents. However, this information can be vital for accurate classification. Therefore, further investigation is needed, and incorporating layout-based approaches such as LayoutLM [98] would be an exciting avenue to explore. In addition, it would be valuable to explore how to incorporate human feedback into the models to enhance their accuracy and address the limitations of automatic classification. Lastly, it is advised to broaden the experiment to additional tasks, such as information extraction through question answering, further to assess the capabilities of the models and products. These investigations can provide valuable insights into the generalizability and adaptability of document classification models and solutions.

Through further investigation and exploration of the mentioned areas, researchers can advance the field of document classification and improve the practical applications of the current models and solutions. These recommendations can deepen the understanding of document classification and similar tasks and increase the efficacy of this industry's current models and solutions.

References

[1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, p. 1–47, mar 2002. [Online]. Available: https://doi.org/10.1145/505282.505283

[2] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, and L. He, "A survey on text classification: From shallow to deep learning," *arXiv preprint arXiv:2008.00364*, 2020.

[3] V. Wagh, S. Khandve, I. Joshi, A. Wani, G. Kale, and R. Joshi, "Comparative study of long document classification," in *TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON)*, 2021, pp. 732–737.

[4] A. A. Soofi and A. Awan, "Classification techniques in machine learning: applications and issues," *Journal of Basic & Applied Sciences*, vol. 13, pp. 459–465, 2017.

[5] P. C. Sen, M. Hajra, and M. Ghosh, "Supervised classification algorithms in machine learning: A survey and review," in *Emerging technology in modelling and graphics*. Springer, 2020, pp. 99–111.

[6] A. Adhikari, A. Ram, R. Tang, and J. Lin, "Docbert: Bert for document classification," *arXiv preprint arXiv:1904.08398*, 2019.

[7] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, 2022.

[8] A. Khan, B. Baharudin, L. H. Lee, and K. Khan, "A review of machine learning algorithms for text-documents classification," *Journal of advances in information technology*, vol. 1, no. 1, pp. 4–20, 2010.

[9] M. Frye and R. H. Schmitt, "Structured data preparation pipeline for machine learning-applications in pro-duction," *17th IMEKO TC*, vol. 10, pp. 241–246, 2020.

[10] B. Bofin, "text classification workflow," https://cloudsek.com/hierarchical-attention-neural-networks-beyond-the-traditional-approaches-for-text-classification/, accessed: 2023-01-05.

[11] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International journal of computer science*, vol. 1, no. 2, pp. 111–117,

2006.

[12] J. Brownlee, "Data preparation for machine learning," 2022.

[13] E. Haddi, X. Liu, and Y. Shi, "The role of text pre-processing in sentiment analysis," *Procedia computer science*, vol. 17, pp. 26–32, 2013.

[14] L. Hickman, S. Thapa, L. Tay, M. Cao, and P. Srinivasan, "Text preprocessing for text mining in organizational research: Review and recommendations," *Organizational Research Methods*, vol. 25, no. 1, pp. 114–146, 2022.

[15] S. Wang, W. Zhou, and C. Jiang, "A survey of word embeddings based on deep learning," *Computing*, vol. 102, no. 3, pp. 717–740, 2020.

[16] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.

[17] R. Ramya, K. Venugopal, S. Iyengar, and L. Patnaik, "Feature extraction and duplicate detection for text mining: A survey," *Global Journal of Computer Science and Technology*, 2017.

[18] E. Elgeldawi, A. Sayed, A. R. Galal, and A. M. Zaki, "Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis," in *Informatics*, vol. 8, no. 4. MDPI, 2021, p. 79.

[19] "machine learning paradigms," https://www.adservio.fr/post/machine-learning-types-benefits, accessed: 2023-01-13.

[20] J. Han, J. Pei, and H. Tong, *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[21] I. H. Sarker, A. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big data*, vol. 7, pp. 1–29, 2020.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[23] I. H. Sarker, Y. B. Abushark, F. Alsolami, and A. I. Khan, "Intrudtree: a machine

learning based cyber security intrusion detection model," *Symmetry*, vol. 12, no. 5, p. 754, 2020.

[24] H. Liu and H. Motoda, *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media, 1998, vol. 453.

[25] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine learning: algorithms and applications*. Crc Press, 2016.

[26] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[28] U. Naseem, I. Razzak, S. K. Khan, and M. Prasad, "A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models," *Transactions on Asian and Low-Resource Language Information Processing*, vol. 20, no. 5, pp. 1–35, 2021.

[29] J. A. Balazs and J. D. Velásquez, "Opinion mining and information fusion: a survey," *Information Fusion*, vol. 27, pp. 95–110, 2016.

[30] S. González-Bailón and G. Paltoglou, "Signals of public opinion in online communication: A comparison of methods and data sources," *The ANNALS of the American Academy of Political and Social Science*, vol. 659, no. 1, pp. 95–107, 2015.

[31] G. C. Banks, H. M. Woznyj, R. S. Wesslen, and R. L. Ross, "A review of best practice recommendations for text analysis in r (and a user-friendly app)," *Journal of Business and Psychology*, vol. 33, p. 445–459, 2018. [Online]. Available: https://doi.org/10.1007/s10869-017-9528-3

[32] Y. Mejova and P. Srinivasan, "Exploring feature definition and selection for sentiment classifiers," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 5, no. 1, 2011, pp. 546–549.

[33] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 519–528.

[34] V. Balakrishnan and L.-Y. Ethel, "Stemming and lemmatization: A comparison of retrieval performances," *Lecture Notes on Software Engineering*, vol. 2, pp. 262–267, 01 2014.

[35] C. C. Aggarwal, *Machine learning for text*. Springer, 2018, vol. 848.

[36] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[37] S. Qaiser and R. Ali, "Text mining: use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018.

[38] S. T. Dumais *et al.*, "Latent semantic analysis," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, no. 1, pp. 188–230, 2004.

[39] Z. Harris, "Distributional structure.(j. katz, ed.) word journal of the international linguistic association, 10 (23), 146-162," 1954.

[40] T. Hofmann, "Probabilistic latent semantic analysis," *arXiv preprint arXiv:1301.6705*, 2013.

[41] E. Altszyler, M. Sigman, S. Ribeiro, and D. F. Slezak, "Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database," *arXiv preprint arXiv:1610.01520*, 2016.

[42] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," *Advances in neural information processing systems*, vol. 29, 2016.

[43] D. Dessì, D. R. Recupero, and H. Sack, "An assessment of deep learning models and word embeddings for toxicity detection within online textual comments," *Electronics*, vol. 10, no. 7, p. 779, 2021.

[44] E. M. Dharma, F. L. Gaol, H. Warnars, and B. Soewito, "The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 2, p. 31, 2022.

[45] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[46] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[47] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv e-prints*, p. arXiv:1802.05365, Feb. 2018.

[48] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

[49] S. Raschka, "Naive bayes and text classification i-introduction and theory," *arXiv preprint arXiv:1410.5329*, 2014.

[50] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[51] D. Berrar, "Bayes' theorem and naive bayes classifier," *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, vol. 403, p. 412, 2018.

[52] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, no. 1. Madison, WI, 1998, pp. 41–48.

[53] K. Shah, H. Patel, D. Sanghvi, and M. Shah, "A comparative analysis of logistic regression, random forest and knn models for the text classification," *Augmented Human Research*, vol. 5, pp. 1–16, 2020.

[54] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and*

ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings. Springer, 2003, pp. 986–996.

[55] P. Soucy and G. W. Mineau, "A simple knn algorithm for text categorization," in *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001, pp. 647–648.

[56] S. Jiang, G. Pang, M. Wu, and L. Kuang, "An improved k-nearest-neighbor algorithm for text categorization," *Expert Systems with Applications*, vol. 39, no. 1, pp. 1503–1509, 2012.

[57] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.

[58] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings*. Springer, 2005, pp. 137–142.

[59] S. Raschka, *Python machine learning*. Packt publishing ltd, 2015.

[60] S. Indra, L. Wikarsa, and R. Turang, "Using logistic regression method to classify tweets into the selected topics," in *2016 international conference on advanced computer science and information systems (icacsis)*. IEEE, 2016, pp. 385–390.

[61] A. DeMaris, "A tutorial in logistic regression," *Journal of Marriage and the Family*, pp. 956–968, 1995.

[62] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[63] V. Jakkula, "Tutorial on support vector machine (svm)," *School of EECS, Washington State University*, vol. 37, no. 2.5, p. 3, 2006.

[64] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[65] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial,"

Computer, vol. 29, no. 3, pp. 31–44, 1996.

[66] "Importance of artificial neural networks in artificial intelligence," https://www.turing.com/kb/importance-of-artificial-neural-networks-in-artificial-intelligence, accessed: 2023-02-07.

[67] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[68] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[69] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[70] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[71] F. Shahid, A. Zameer, and M. Muneeb, "Predictions for covid-19 with deep learning models of lstm, gru and bi-lstm," *Chaos, Solitons & Fractals*, vol. 140, p. 110212, 2020.

[72] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[73] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[74] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[75] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," in *Proceedings of the 55th Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers), 2017, pp. 562–570.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[77] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," OpenAI, Tech. Rep., 2018.

[78] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[79] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[80] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[81] C. F. P. Bureau, "Consumer complaints report 2021," https://files.consumerfinance.gov/f/documents/cfpb_2021-consumer-response-annual-report_2022-03.pdf, accessed: 2023-01-09.

[82] S. Li, "Multi-class text classification with lstm," https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17, accessed: 2023-01-25.

[83] E. Alzahrani and L. Jololian, "How different text-preprocessing techniques using the bert model affect the gender profiling of authors," *arXiv preprint arXiv:2109.13890*, 2021.

[84] A. Kurniasih and L. P. Manik, "On the role of text preprocessing in bert embedding-based dnns for classifying informal texts," *Neuron*, vol. 1024, no. 512, p. 256, 2022.

[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[86] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[87] "Microsoft azure," https://language.cognitive.azure.com/, accessed: 2023-02-28.

[88] "Aws comprehend," https://aws.amazon.com/de/comprehend/, accessed: 2023-02-28.

[89] "Openai," https://openai.com/, accessed: 2023-02-28.

[90] R. Dale, "Gpt-3: What's it good for?" *Natural Language Engineering*, vol. 27, no. 1, pp. 113–118, 2021.

[91] "Levity ai," https://levity.ai/, accessed: 2023-02-28.

[92] "Planet ai," https://planet-ai.de/, accessed: 2023-02-28.

[93] "Google vertex ai," https://cloud.google.com/vertex-ai?hl=de, accessed: 2023-02-28.

[94] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining & knowledge management process*, vol. 5, no. 2, p. 1, 2015.

[95] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

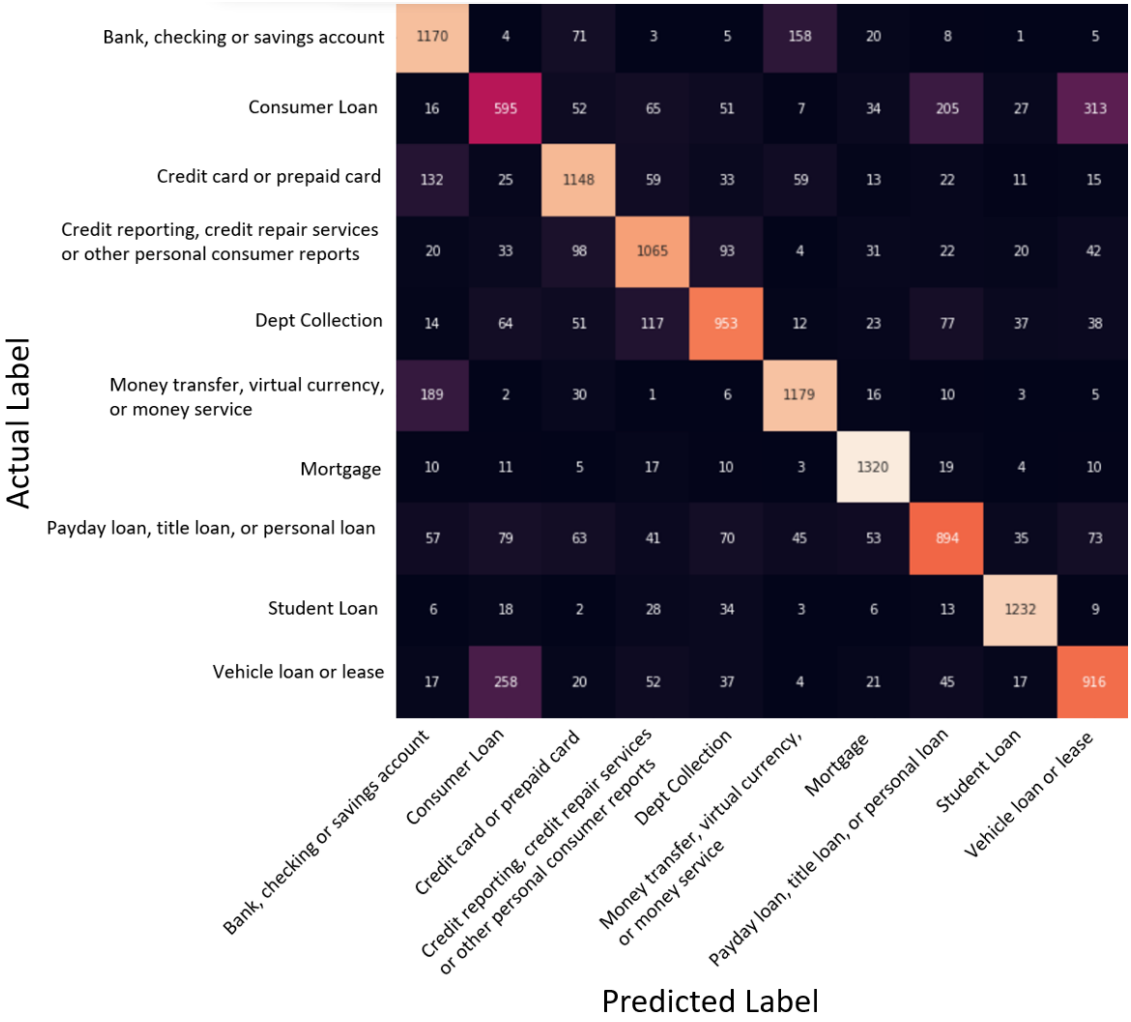
[96] J. Mohajon, "confusion matrix," https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826, accessed: 2023-02-07.

[97] "sagemaker workflow," https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html#deploy-model-steps-byom, accessed: 2023-03-06.

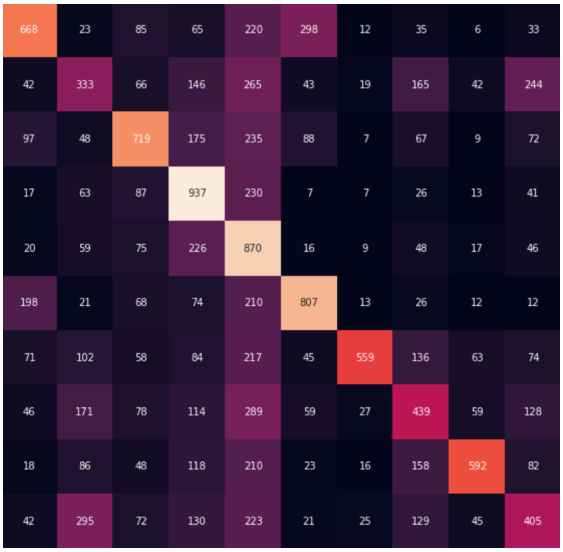
[98] Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, "Layoutlm: Pre-training of text and layout for document image understanding," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1192–1200.

Appendices

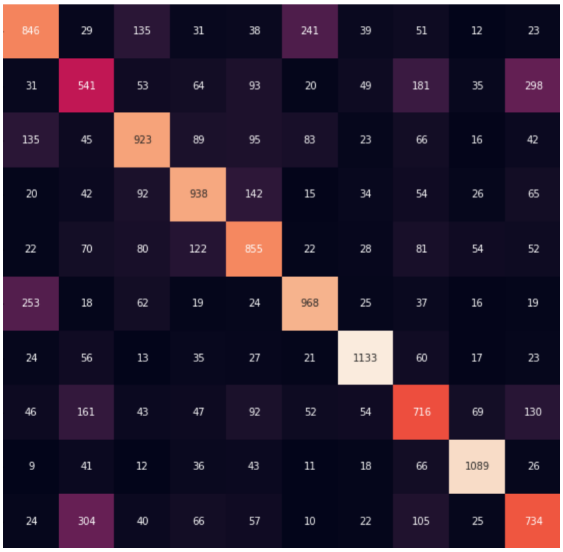
A. Confusion Matrices



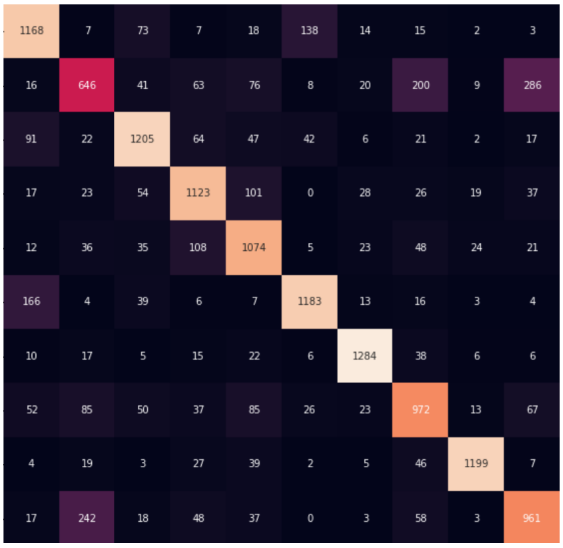
(a) NBC



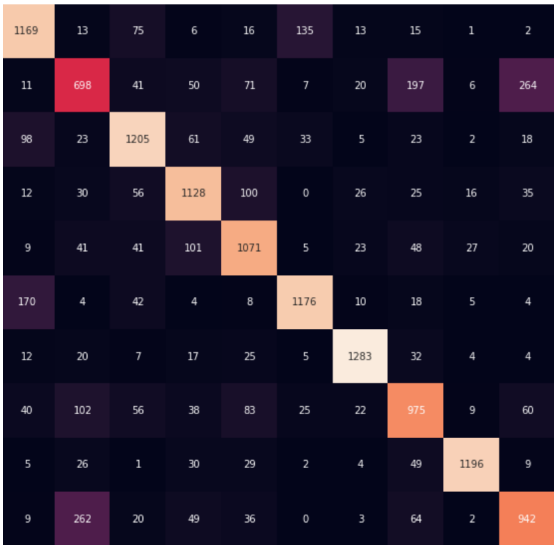
(b) KNN



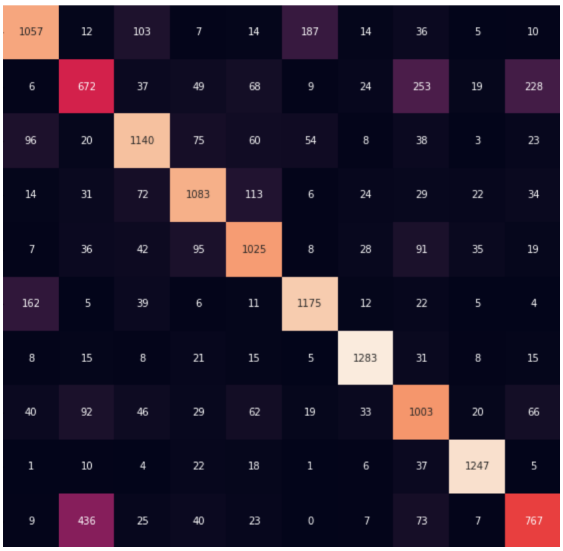
(c) Decision Tree



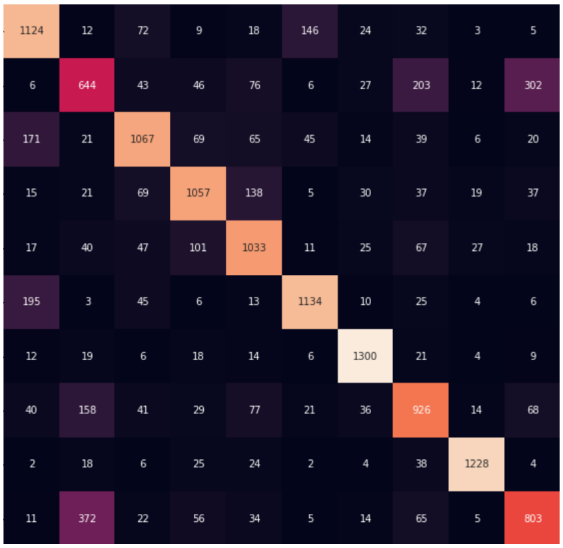
(d) Logistic Regression



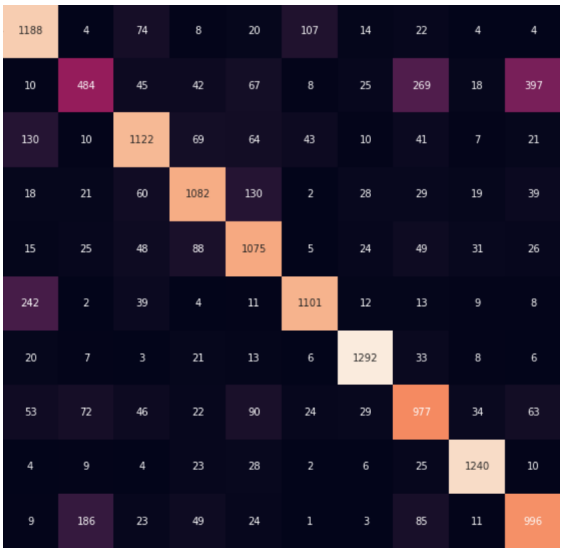
(e) SVM



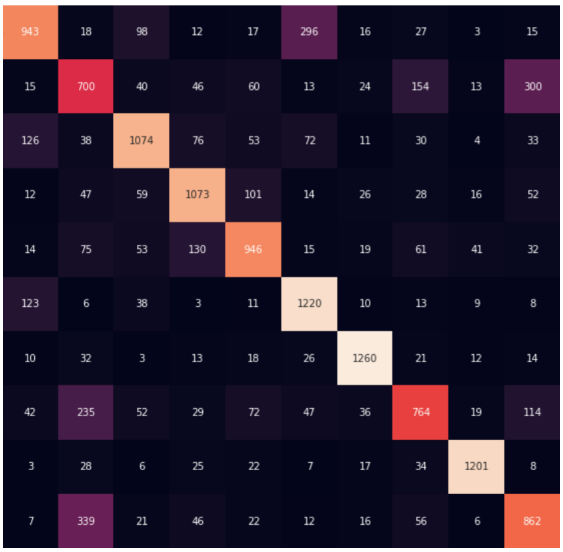
(f) LSTM



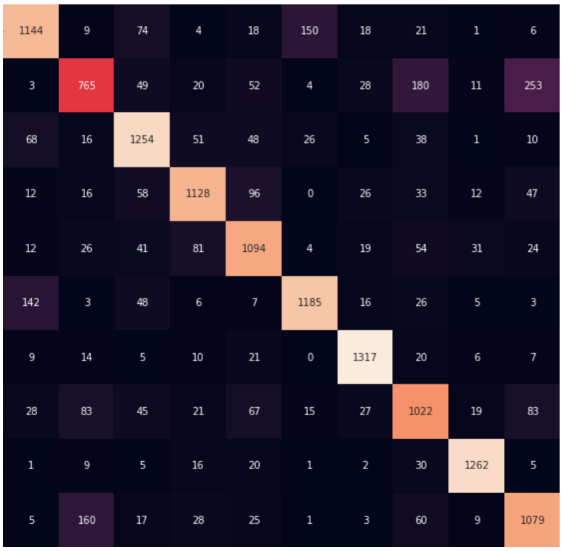
(g) Bi-LSTM



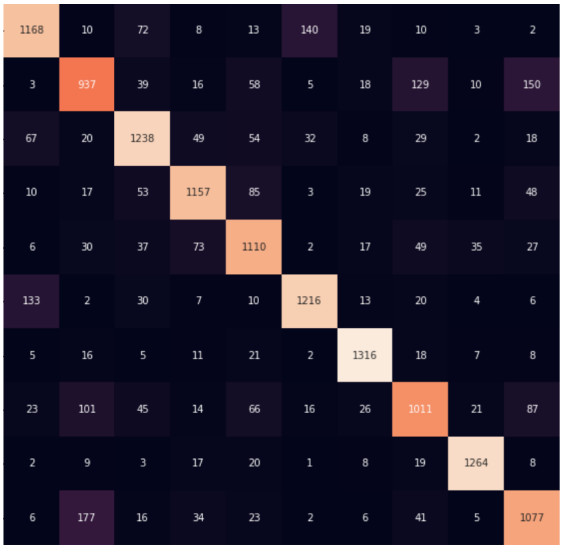
(h) GRU



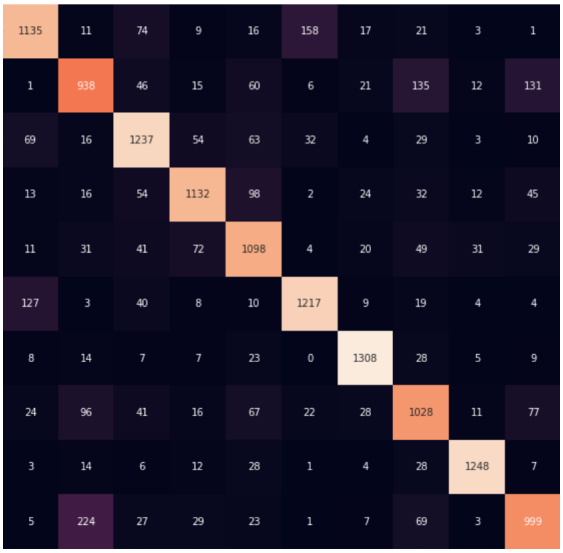
(i) CNN



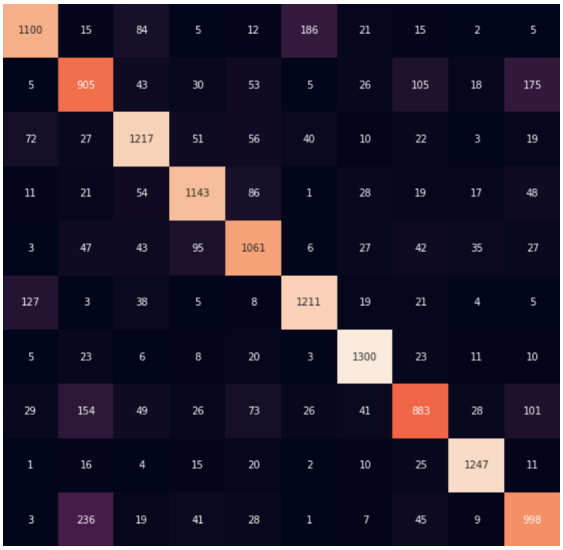
(j) BERT



(k) RoBERTa

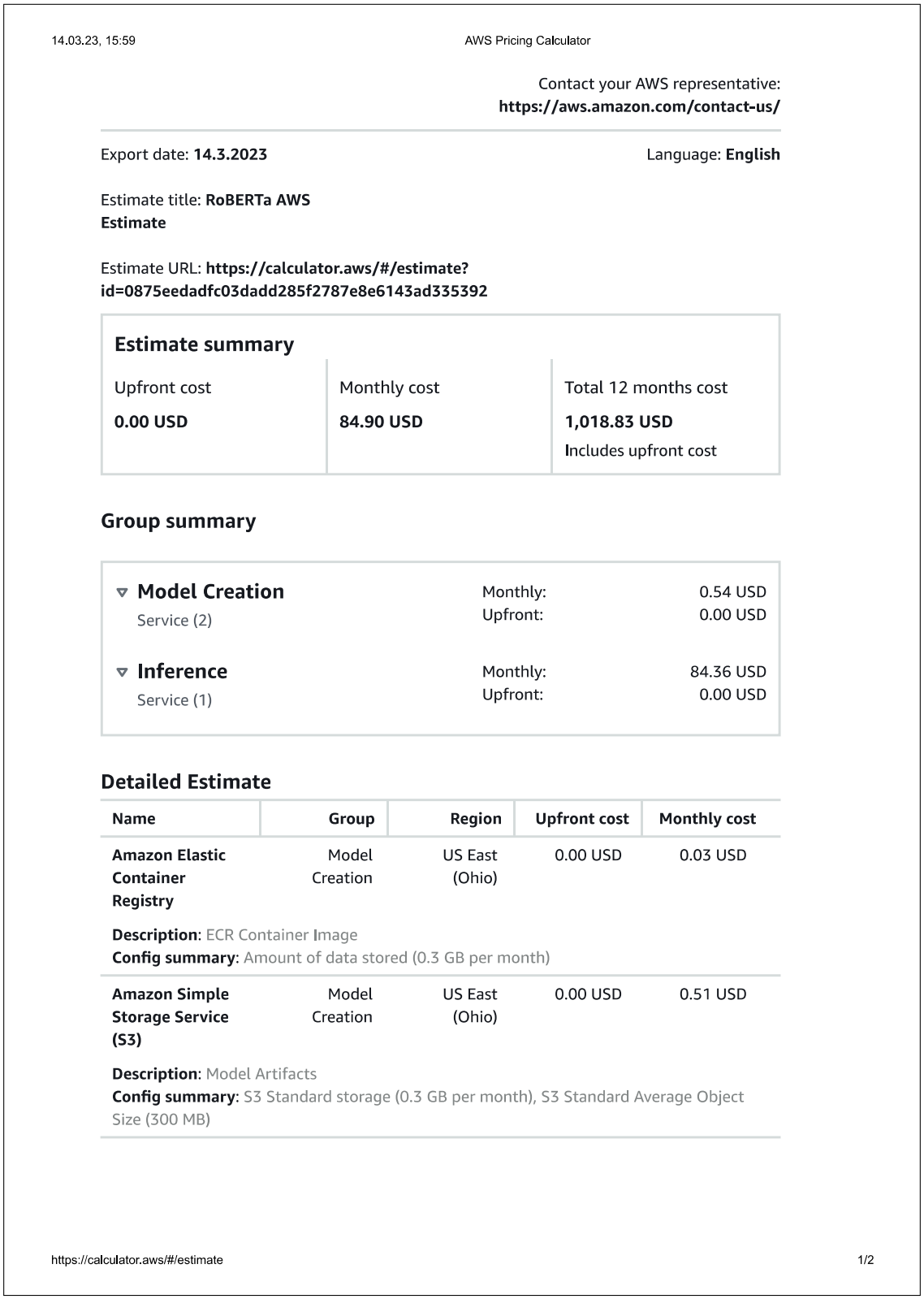


(l) XLNet



(m) GPT-2

B. Amazon SageMaker Costs Estimation



14.03.23, 15:59 AWS Pricing Calculator

Contact your AWS representative:
https://aws.amazon.com/contact-us/

Export date: **14.3.2023** Language: **English**

Estimate title: **RoBERTa AWS Estimate**

Estimate URL: **https://calculator.aws/#/estimate?id=0875eedadfc03dadd285f2787e8e6143ad335392**

Estimate summary

Upfront cost	Monthly cost	Total 12 months cost
0.00 USD	**84.90 USD**	**1,018.83 USD** Includes upfront cost

Group summary

Model Creation Service (2)	Monthly: Upfront:	0.54 USD 0.00 USD
Inference Service (1)	Monthly: Upfront:	84.36 USD 0.00 USD

Detailed Estimate

Name	Group	Region	Upfront cost	Monthly cost
Amazon Elastic Container Registry	Model Creation	US East (Ohio)	0.00 USD	0.03 USD
Description: ECR Container Image **Config summary**: Amount of data stored (0.3 GB per month)				
Amazon Simple Storage Service (S3)	Model Creation	US East (Ohio)	0.00 USD	0.51 USD
Description: Model Artifacts **Config summary**: S3 Standard storage (0.3 GB per month), S3 Standard Average Object Size (300 MB)				

https://calculator.aws/#/estimate 1/2

Figure 8: AWS Pricing Estimation[1]Page 1

[1] `https://calculator.aws/#/estimate?id=bcab204cad4e5f9a669948f2f2cb128f00f45ead`

Amazon SageMaker	Inference	US East (Ohio)	0.00 USD	84.36 USD

Description: Model Hosting + Real-Time Inference
Config summary: Storage (General Purpose SSD (gp2)), Instance name (ml.m5.large), Instance name (ml.m5.large), Number of models deployed (1), Number of models per endpoint (1), Number of instances per endpoint (1), Endpoint hour(s) per day (24), Endpoint day(s) per month (30), Data Processed IN (100000 MB), Number of Model Monitor jobs per month (0), Number of Model Monitor instances per job (0), Hour(s) per Model Monitor instance per job (0)

Acknowledgement

AWS Pricing Calculator provides only an estimate of your AWS fees and doesn't include any taxes that might apply. Your actual fees depend on a variety of factors, including your actual usage of AWS services. **Learn more**.

Figure 9: AWS Pricing Estimation Page 2

Acronyms

API Application Programming Interface. 35, 37, 42, 45, 46

ASCII American Standard Code for Information Interchange. 14, 34

AWS Amazon Web Services. 41, 47, 51, 53, 68

BERT Bidirectional Encoder Representations from Transformers. 6, 28, 29, 34, 35, 44, 52, 67

Bi-LSTM Bidirectional Long Short-Term Memory. 26, 27, 44, 50, 52, 66

BOW Bag-of-Words. 16, 18, 35

CBOW Continuous Bag of Words. 19

CFPB Consumer Financial Protection Bureau. 30, 31, 52

CNN Convolutional Neural Network. 20, 25–29, 35, 44, 50, 52, 66

CPU Central Processing Unit. 46

DPCNN Deep Pyramid Convolutional Neural Network. 28

ECR Elastic Container Registry. 41, 48

ELMo Embedding from Language Models. 20

FN False Negative. 39

FOSS Free Open-Source Software. 5, 28, 30, 38, 40–46, 50, 52–54

FP False Positive. 39

GloVe Global Vectors for word representation. 19, 20

GPT Generative Pre-trained Transformer. 28, 29, 35–37, 44–54, 67

GPU Graphics Processing Unit. 43, 45, 46, 50

GRU Gated Recurrent Unit. 26, 27, 35, 44, 52, 66

IDF Inverse Document Frequency. 17

KNN K-Nearest Neighbor. 20, 22, 23, 35, 44, 50, 52, 65

LSA Latent Semantic Analysis. 18, 35

LSTM Long Short-Term Memory. 26, 27, 35, 44, 50, 52, 66

NBC Naive Bayes Classifier. 20–22, 35, 44, 50, 52, 64

NLP Natural Language Processing. 5, 6, 18–20, 25–29, 36

OCR Optical Character Recognition. 8

PLM Transformer-based Pre-trained Language Model. 6, 28, 29, 34, 50–53

RNN Recurrent Neural Network. 20, 25–29, 44, 50

RoBERTa Robustly Optimized BERT Pre-training Approach. 29, 34, 35, 44–53, 67

SVD Singular Value Decomposition. 18

SVM Support Vector Machine. 20, 24, 25, 35, 44, 50, 52, 65

TF Term Frequency. 17

TF-IDF Term Frequency-Inverse Document Frequency. 17, 18, 35

TN True Negative. 39

TP True Positive. 39